FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Test Automation Best Practices with Test Process Lines

A Case Study on Classified Ads Platforms

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Christoph Putz, BSc
Matrikelnummer 1060602

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dr. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dr.techn. Dietmar Winkler

Wien, 30. September 2019

Christoph Putz	Stefan Biffl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Test Automation Best Practices with Test Process Lines

A Case Study on Classified Ads Platforms

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Christoph Putz, BSc
Registration Number 1060602

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Dr. Stefan Biffl
Assistance: Dipl.-Ing. Dr.techn. Dietmar Winkler

Vienna, 30th September, 2019

Christoph Putz | Stefan Biffl

Erklärung zur Verfassung der Arbeit

Christoph Putz, BSc
Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. September 2019

Christoph Putz

Danksagung

Ich möchte mich zuerst bei Dr. Stefan Biffl bedanken für das Ermöglichen dieser Masterarbeit, seinen Ideen zur Verbesserung und seiner konstruktiven Kritik. Neben ihm möchte ich mich auch ganz herzlich bei Dr. Dietmar Winkler bedanken, der sich meine Arbeit mehrmals durchgeschaut und immer wieder mit hilfreichen Kommentaren meine Probleme gelöst hat. Danach möchte ich mich bei Willhaben bedanken, die mir durch Einsicht in die internen Prozesse und Tests, die sie tagtäglich durchführen, die Analysen ermöglicht haben. Außerdem möchte ich mich bei allen Teilnehmern des Surveys als auch des Interviews bedanken, da es nicht selbstverständlich ist, sich Zeit für solche Befragungen zu nehmen. In diesem Zuge möchte ich auch die Unterstützung der Schibsted betonen, die solche Projekte immer unterstützen und fördern. Außerdem möchte ich mich bei meiner Familie für das Vertrauen bedanken, dass sie mir immer geschenkt haben und auch die Ermöglichung des Studiums der Informatik im Allgemeinen. Am Schluss möchte ich mich noch besonders bei meiner Freundin bedanken, die die Idee für diese Arbeit geliefert hat und mich immer unterstützt hat, wenn einmal weniger Motivation vorhanden war.

Acknowledgements

I would first like to thank Dr. Stefan Biffl for making this master thesis possible, his ideas for improvement and his constructive criticism. In addition to him, I would also like to thank Dr. Dietmar Winkler, who has looked through my work several times and repeatedly solved my problems with helpful comments. Then I would like to thank Willhaben, who made my analyses possible by giving me insight into the internal processes and tests they carry out on a daily basis. I would also like to thank all the participants of the survey and the interview, as it is not a matter of course to take time for such surveys. In this context, I would also like to emphasize the support of Schibsted, who always support and promote such projects. I would also like to thank my family for the trust they have always placed in me and for enabling me to study computer science in general. Finally, I would like to thank my girlfriend in particular, who gave me the idea for this work and always supported me when there was less motivation.

Kurzfassung

In den letzten Jahren sind agile Entwicklungsmethoden immer beliebter geworden und viele Unternehmen haben auf SCRUM und KANBAN umgestellt. In diesem Zusammenhang wird auch die Testautomatisierung immer wichtiger, da Softwaretests immer früher im Entwicklungsprozess eingebunden werden. Zu diesen Softwaretests gehören auch *End-to-End (E2E)* Tests, die dafür eingesetzt werden um Softwaredefekte früher zu entdecken.

Diese Arbeit zielt darauf ab, einen Maßnahmenkatalog für den Testprozess, sowie Best Practices für die Testautomatisierung zu generieren. Außerdem soll anhand eines Praxisbespiels veranschaulicht werden wie zurzeit Testautomatisierung in einem Konzern funktioniert und wie diese verbessert werden kann.

In dieser Arbeit werden verschiedene Methoden für die Testautomatisierung vorgestellt und der Test Prozess von Online- Marktplätzen untersucht. Der Testprozess wird mit SPICE und mit dem Einsatz einer Software Prozess Linie verbessert. Als Use Case dient hier das Unternehmen Willhaben. Es wird mithilfe einer Umfrage und strukturierten Interviews festgestellt, inwieweit der Testprozess in den Unternehmen mit den Erwartungen der Mitarbeiter übereinstimmt. Die Methoden für die Testautomatisierung werden analysiert und bewertet um dann für die Best Practices gesammelt zu werden.

Mit einer quantitativen Umfrage und 10 Interviewpartnern wurde ein guter Überblick über die verschiedenen Testprozesse in den verschiedenen Organisationen im Bereich der Onlinemarktplätze geschaffen. Hier kann man mit den Best Practices und SPICE zum Bewerten des vorhandenen Test Prozesses gut ansetzen. Es wurde außerdem klar, dass viele Unternehmen nur teilweise auf E2E Tests setzen, hier ist also noch Verbesserungspotential vorhanden. Dass eine Software Prozess Linie für einen allgemeinen Testprozess in einem Unternehmen sinnvoll ist, wird durch die verschiedenen Aussagen der Interviewpartner unterstrichen.

Die Resultate dieser Arbeit unterstreichen die Wichtigkeit eines gut durchdachten Testprozesses, der dokumentiert und schriftlich festgehalten ist. Das Bewerten des Testprozesses durch SPICE erschafft sofort Verbesserungspotential, welches durch verschiedenste Maßnahmen umgesetzt werden kann. Best Practices sollten, sofern sie für das Unternehmen umsetzbar sind, eingesetzt werden. Die Auswirkungen auf Firmen mit anderen Beschäftigungsfeldern können anders sein als im betrachteten Anwendungsfall, aber dies wird nicht in dieser Arbeit bearbeitet.

Abstract

In recent years, agile development methods have become increasingly popular and many companies have switched to SCRUM and KANBAN. In this context, test automation becomes more important as software testing is getting involved earlier in the development process. These software tests also include *End-to-End (E2E)* testing, which is also used to detect software defects earlier.

The objectives of this thesis are to develop a catalogue of improvements for the test process as well as to generate best practices for test automation. In addition, a practical example will be used to illustrate how test automation is currently carried out in a corporation and how it can be improved.

In this work, several test automation best practices will be presented and the testing process of online marketplaces will be explored and enhanced with SPICE and with the usage of a software process line. The use case here is the company Willhaben. With the support of a survey and structured interviews it will be analysed how the test process aligns according to the expectations of the employees. The test automation methods are analysed and evaluated to be collected for best practices.

With a quantitative survey and 10 interview partners, a good overview of the various test processes in the various organizations in the area of online marketplaces was created. The best practices and SPICE for evaluating the existing test process are used here. It also became clear that many companies only partially rely on E2E tests, so there is still room for improvement. The fact that a software process line makes sense for a general test process in a company is underlined by the various statements of the interview partners.

The results of this work underline the importance of a well thought-out testing process, documented and written down. The evaluation of the test process by SPICE immediately creates potential for improvement, which can be implemented by various measures. Best practices should be used if they are feasible for the company. The effects on companies with other fields of employment can be different than in the considered use case, but this is not part of this work.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 2

2 Software Testing **7**
2.1 Software Test Process 7
2.2 Software Testing Techniques 13
2.3 Test Frameworks 25

3 Software Process Improvement with Process Lines **43**
3.1 Software Process Line 55

4 Illustrative Use Case **59**

5 Research Questions **63**
5.1 Best Practices for Software Testing 63
5.2 Specific Requirements E2E-Tests 64
5.3 Improve an Software Test Process Approach 65

6 Solution Approach **67**

7 Initial Analysis at Willhaben **71**
7.1 Current State of Practice 71
7.2 SPICE Assessment of Selected Process Areas 73
7.3 Test Process Improvements 83
7.4 Software Process Line 85

8 State of Practice Survey **87**
8.1 Survey Design 87

8.2 Survey Results 89

9 Expert Interviews **109**
9.1 Expert Interviews Structure 109
9.2 Results and Analysis of Expert Interviews 111

10 Improving Test Automation at Willhaben **115**
10.1 Test Process Analysis 115
10.2 Improved Test Automation Process 116
10.3 Evaluation of E2E Tests 120

11 Discussion **123**
11.1 Best practices (RQ.1) 123
11.2 Requirements for E2E tests (RQ.2) 125
11.3 Catalogue of Improvements 127
11.4 Improved Test Process (RQ.3) 129

12 Limitations & Threats to Validity **133**
12.1 External Threats 133
12.2 Internal Threats 134

13 Conclusion **137**
13.1 Lessons Learned and Cost Benefit Analysis 137
13.2 Future Work 139

List of Figures **141**

List of Tables **143**

Bibliography **145**

Online References **153**

CHAPTER 1

Introduction

The problems that software testing and in particular test automation has to solve in companies that consist of several subcontractors are diverse. On the one hand, different test processes are used in the companies as soon as there is no uniform solution from the parent company. On the other hand, different solutions are sought and found for the same problems.

1.1 Motivation

In this work we investigate best practices in *test automation (TA)* with the focus on classifieds, or marketplaces in the world wide web like for example Willhaben in Austria. Software testing is important when developing a new software or implement new features into an existing program [21, 22, 25]. There are a lot of different papers or books regarding software testing in general [14, 21, 22] or testing in a specific area like regression testing and unit testing [7, 13, 15] for example. Other researches cover the testing methods and how those can be achieved through different testing approaches [4, 9]. There are also researches on how much value can be generated, considering software testing a software project and how much the implementation of such testing would cost [14, 26]

But there are not that many papers or books about TA in marketplaces and even fewer when considering best practices on how to handle TA in a global way. Therefore this work should provide an insight on how best practices could look like and how TA is perceived in companies, regarding the view of developers, testers and management persons as well. There are papers on automated software testing such as [16, 17], but they focus on TA in general or are specifically for one project. Others focus on specific automation tools like Struts [27] or Selenium in a large TA project [28], which was done in 2009 and is specifically used to get insights into the history of TA. Selenium will be explained later in state of the art as it is still used in many projects and has been developed further to be up to date [10].

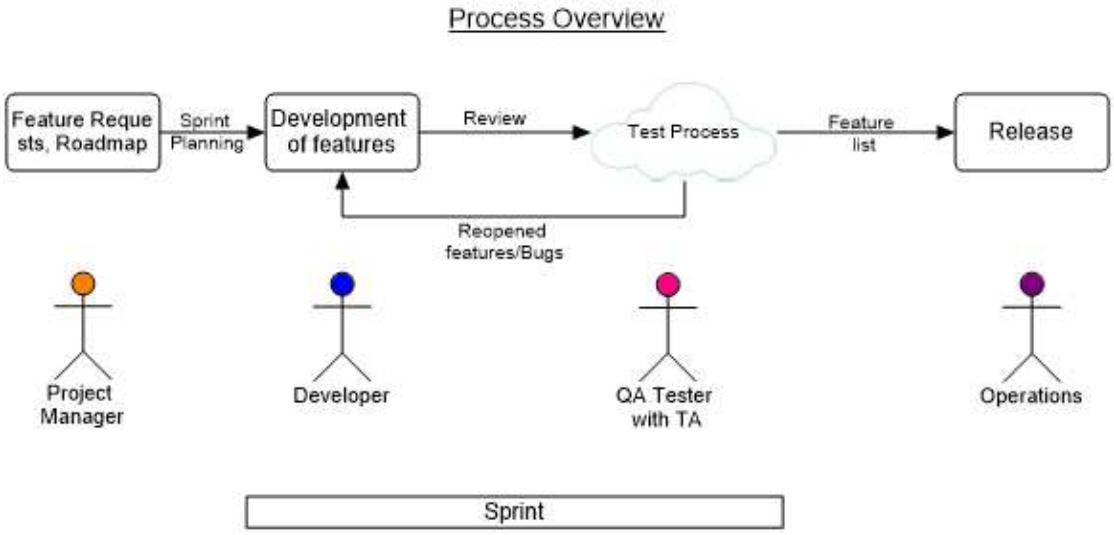

Figure 1.1: Release Process at Willhaben

Another focus is on finding a *Software Process Line (SPrL)* for software testing for a globally operating company, as there are just a few papers on software process lines in general, such as [65, 67, 68, 69]. All of those researches try to optimize a process in companies with various software process lines, so that the software developers don't have to spend too much time to find the optimal software process for every development task, but they are not specifically designed for software testing and how to implement a SPrL for this test process.

To define, improve and adapt a test process for a marketplace like Willhaben is another objective of this work. There are different test process improvement approaches which can be used such as *Software Process Improvement and Capability Determination (SPICE)*, *Capability Maturity Model Integration (CMMI)* or *Plan-Do-Check-Act (PDCA)*, which are further described in section 3. In [73, 77, 82] the different approaches are described in more detail. Those books are the basis for the standards and describe how the process have to be applied to improve a software process. This will be used in this work to improve the test process.

In figure 1.1 the release process of Willhaben is shown. The test process is the part which is important for this work and the adaptation and improvement is one of the problems that has to be solved. There are different players in the current process and the involvement of them is shown as well.

1.2 Problem Statement

Testing is a time-consuming task and requires many resources which, if not used for testing, could be used for other investments. Manual testers often need to perform

regression testing for each release, which means running the same tests over and over again to prevent recurring errors from entering the *System Under Test (SUT)*. These manual testers are, of course, a cost factor, but they are also specialists in finding bugs in the SUT. Instead, they should be used to find bugs in newly developed features or edge cases. These are hard to find and take a lot of time to reproduce. This is where test automation comes in. TA could reduce the workload of regression testing for manual testers and help them not to spend too much time on recurring errors and let them find newly introduced errors. Therefore, this work compares the strengths and weaknesses of manual testers and TA against each other and evaluates best practices on how to use TA most effectively in an agile organization.

It's these agile methods that exacerbate the problem, as the release cycles and feature broadcasts are either non-existent because of continuous deployment or are performed at such a high rate that the testers constantly perform only regression testing. They don't have time to test functions thoroughly because they are under too much pressure.

The difference between continuous delivery and continuous deployment is the act of deploying code changes to production [1]. In continuous delivery deploying new code is done manually when the tests of the SUT are done and the release can be handled by the company. In continuous deployment all of that is done automatically as every change of code, that passes the automated tests, that are in place, is deployed to production [1]. This is of course stressful for manual testers as they have no control over what is deployed on to production and the testing starts when the change is in production. Therefore the companies often decide to use software testing to decrease the amount of workload a manual tester has to do. In 1.2 the ideal testing pyramid is shown. This figure has often little in common with the reality as many companies have the exact opposite as stated by [48, 47].

As most leading technology companies use agile software development practices as stated in [42], the need for TA is steadily increasing and therefore the need for testers that are more proficient in software development is increasing as well. This work will look at the problems that software testers have and how these problems could partially be solved by TA. The change from *Test-Driven-Development (TDD)* to *Behaviour-Driven-Development (BDD)* is another challenge not only the testers have to face but also the developers. This work has the objective to find best practices on how to avoid bad testing and failing when changing from manual testing to automated software testing. It should benefit large companies and every tester that is affected by the change in his/her company from going away from a traditional waterfall development to a more agile way. This also includes moving from a monthly deployment to maybe a weekly or even more drastic to a *Continuous Deployment (CD)* schedule.

Additionally, software testing and the test process are still immature in many companies as stated by [58]. Those processes are usually done in an ad-hoc fashion. This can lead

[1] Continuous delivery, continuous deployment; puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff Accessed: 2018-08-29

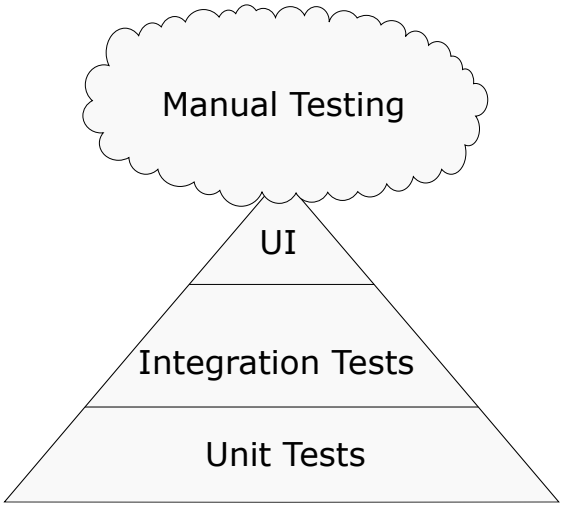

Figure 1.2: Ideal testing pyramid

to negative results and failure in a project, as testing isn't conducted in the right way. These failures can lead to cost explosions, not detecting defects in the right way, or not communicating and reporting them correctly [58]. Therefore this work tries to find best practices how a test process should be like and compare the theoretic approach to the approach taken by companies from Schibsted in the classifieds spectrum.

Software testing is also a cost factor for every company that develops software or provides *Software as a Service (SaaS)*, as the software has to be tested and it should be done as good as possible [21, 58]. Testing could even have saved life on some occasions, as stated by [21], if it had been done with more insights, as we have now. As stated by [59], in 2013 half of the development time of an average program was taken by software testing and $312 billion were spent on detecting and fixing software bugs. This work's scope is also set on how to reduce the costs of software testing, while maintaining the quality or even increase the quality of the researched programs. This will be done by a catalogue of improvements that can be used to find cases where a re-evaluation should have been done. As stated by [21] the worst case scenario is to think that no testing would help, as this increases the costs even more, when a newly developed program is a complete failure or needs so much refactoring to work, that the costs explode. This can be seen in figure 1.3. The cost at release isn't zero, it's exactly the opposite, as it isn't calculable.

The graph shows how much an undetected bug can potentially cost and that it's better to find it as early as possible. This can be done in various ways, which are described later in 2.

The overall goal is therefore to find best practices to lower costs, save time and have a better understanding for TA in general, as well as to achieve a test process which is usable in a wide scale, so that it can support the development of TA and with a specific

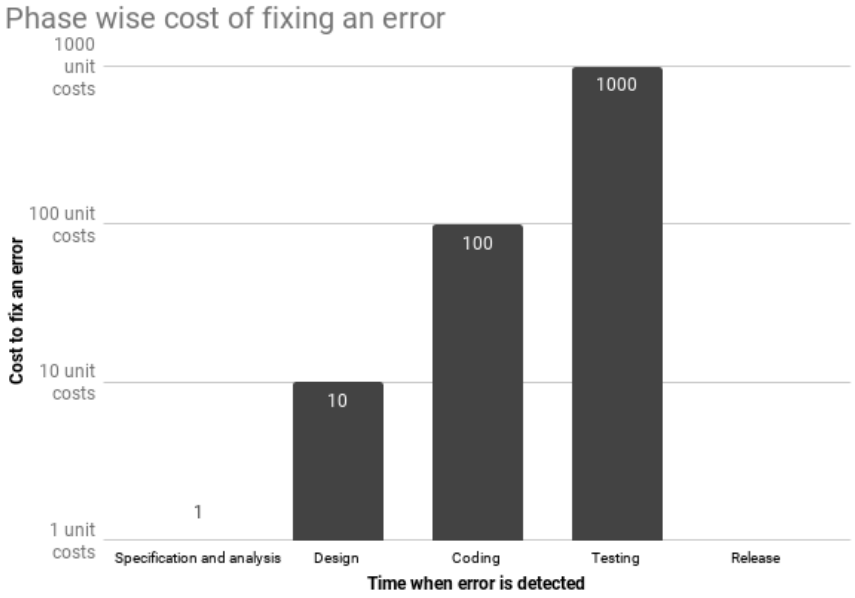

Figure 1.3: Costs of fixing an error from [21]

focus on E2E tests. The SPrL will be checked if it provides benefits that outweigh the initial setup costs.

The research goal of this work is to gain a better understanding of how best practices are best applied and whether they have a positive impact on the cost of testing. It also aims to find out what effect these best practices have on an agile organization. Second, we will investigate which conditions must be met to use E2E tests effectively and how these tests can support manual testers. In addition, it will be investigated how test automation influences the costs of software development and whether the test process changes significantly due to the use of test automation. The effects on the manual testers and also the developers are an additional matter to be investigated. Furthermore, we will check how a test process can look like and how to change an existing test process to get the maximum out of it.

Finding best practices for testing with E2E tests as well as with other test types is one of the goals of this work. These best practices and an analysis of the test methods used in Willhaben are intended to identify specific requirements for E2E tests. The second goal is to analyze and improve a testing process using an application case, in this case Willhaben, where to find out where a company can develop when applying a defined testing process. The potential for improvement is then presented as a catalogue of actions.

CHAPTER 2

Software Testing

In this chapter the state of the art is shown and explained. It will start with the general software test process, how to include agile processes in software testing. After that we present software testing with a short overview of the software development life-cycle at the beginning, which different approaches to software testing can use and how this is done. After that we discuss how the test processes are best used. This includes the different approaches to software testing like unit, integration, functional testing and end-to-end testing. Subsequently some of the mainly used different testing frameworks are explained and shown.

Furthermore, the advantages and disadvantages of end to end tests in comparison to manual testing will be described and picked up with different sources. Finally specific case studies which were done in the field of software testing will be discussed and compared to the approach in this work. The main programming language for the frameworks that will be looked into will be Java. There are some frameworks, especially for mobile applications that are not specified for one specific language but this will be explained as well in the specific section.

2.1 Software Test Process

In this section we describe and show the different software test processes and how agile processes work with them. A software test process is used in every company, but can differ in many aspects as it depends a lot on the company team structure and the development practices [34, 37]. A high quality process improves the software product significantly and therefore is important to be developed and maintained.

2.1.1 General Test Process

In [34] the authors list different test process models which can improve the test process. As stated the quality assurance tasks take up 50% to 60% of the development effort, that is put into a product. This indicates that the test process behind it should be well structured. The following section 3 explains a software process improvement approach, which focuses on improving a test process in a company.

A test process can and should be used in every software process model such as traditional, such as waterfall, agile, such as SCRUM or Kanban or exploratory. Every one of these software processes involve testing in one way or the other involved, as can be found in [35] or in [36]. In [36] the different phases of SCRUM are explained and how testing is involved in those phases.

A test process consists of the key components [1] test strategy, test plan, test design, test execution and test closure. These five steps of a test process are the rudimentary steps needed to succeed with a test process. As stated in [37] there are additional key requirements for a test process such as a configuration management, a defect tracking tool and the verification of the requirements of the *Application Under Test (AUT)*.

A test process has different key roles that have to be assigned and those are the testing manager, test team leader, tester, test analyst and the independent observer. The testing manager is responsible for the organizational aspects of the test process and has to have insights into the development teams as well, so that these teams keep up the quality with unit and integration tests, that are developed by them. Those tests were described in 2.2.2 and are an essential part of the test process. The testing manager has to think through the test strategy for all the projects. [37]

The test team leader is the lead for running the testing project. He/She has the responsibility to task the testers and test analysts that are in his/her team. The test team leader has to set up the test plan and the test specification. [37]

The tester is the bread and butter of the testing team, and executes and/or develops automated tests, as well as manual tests. He/She has to analyze the test results and document them. He/She is responsible for the data back ups of the test documentation and archival of the test data. [37]

The test analyst is responsible for the design and implementation of the test cases. The analyst can also help the test team lead to generate the specification document. He/She is responsible to prioritize the test cases and has to analyze and identify the specific requirements that have to be verified by the tester. [37]

The independent test observer is outside the test team and provides independent verification of the testing procedures. This includes that observer must ensure that the tester executes, analyses and records the tests and test results according to the test scripts. [37]

[1] Software testing, test process, stakeholders; reqtest.com/testing-blog/the-a-to-z-guide-to-the-software-testing-process/ Accessed 27.12.2018

[2] V-Model; www.softwaretestingclass.com/v-model/ Accessed: 27.12.2018

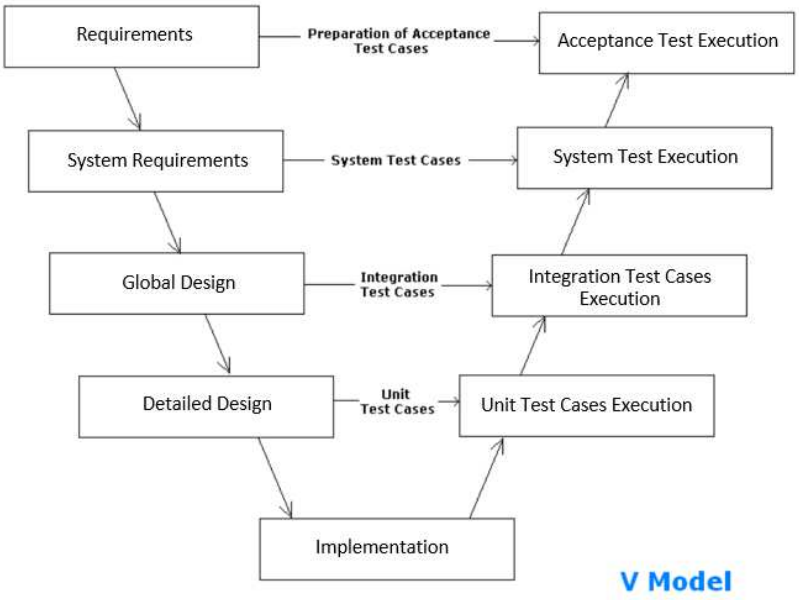

Figure 2.1: V-model of the software development cycle from [2]

There are additional supplementary testing roles, which are described in [37].

In figure 2.1 the V-model which is a software development and testing model, is displayed. As stated by [37] it shows that a plan and preparation for testing is needed early if a software product should be successful. Each of the development phases in the V-model is directly linked to a testing phase which has to be planned and then executed. The V-model is used in a traditional waterfall development process. In projects with an agile development process the test process looks slightly different, as it is more tightly coupled with the development phase and is sometimes even done before development, such as *Test-Driven-Development (TDD)*. As described in [37] each step of the development phase in an agile process can be seen as a "mini-waterfall", which uses every phase of the V-model for the small step. [36]

At the end of the test process is the step to stop testing the AUT. This isn't an easy task as stated by [37], because the question that arises is, when is the perfect time to stop? This is especially critical when a project has to be delivered and can't be delayed because of testing, which shouldn't be the reason to stop testing, as it can potentially harm the AUT. In [37] there are four criteria mentioned, which help to decide if enough testing was done:

- Test requirement coverage: Are the most critical requirements tested or even every requirement?

- Test code coverage: Is the test coverage high enough?
- Test case metric: A metric to decide if enough tests were planned, designed, implemented, executed and passed or failed, and to collect data throughout the test process of the activities and progress
- Defect detection metric: This metric provides an indicator if the majority of defects have been found and solved, but it can't provide full prove of failure free software

The last criteria tries to give confidence that there won't occur major defects, but as stated in different resources, no testing effort can provide that a software is bug free. [57, 44, 37]

An important point that is made by [37] is that test planning and design should be done as early as possible, as it enables to identify and remove risks before they even occur. And if they occur there is a strategy in place to solve them faster and remove them from the AUT. This is also mentioned in [36]. The definition of done in Scrum includes testing from the developers and from the testers, which are included in the Scrum team.

The software development process influences the test process slightly, as it makes a difference if a traditional or an agile process is used, as the planning and execution differs, but in the end, in both of the processes, testing is an essential part, as can be seen in [36, 37].

2.1.2 Software Testing in Agile Development Environments

In this section different agile methods are briefly introduced and the connection between agile environments and software testing is shown. The main agile processes considered here are Scrum and Kanban. Scrum is more important in this work, as it is the agile process that is used at Willhaben as stated in chapter 1.

Scrum

One of the most used agile frameworks used in software development is Scrum. Those software projects are, as stated by [95], mostly complex and large, as Scrum shouldn't be used for small, stable environment. With Scrum a development team is able to react faster to changes in the specifications and environment. Scrum consists of certain parts, such as a sprint, artifacts and a scrum team.

The sprint is a period of time, up to 4 weeks, where a usable and releasable product is developed. In this the output target defined at the beginning cannot be changed. All features that are not finished will be taken over into the next sprint [95].

Known Scrum artifacts are the product backlog, the sprint backlog and the resulting increment. The increment also contains the definition of done that is defined by the Scrum team. The more experienced a team is, the more comprehensive the done criteria are. The product backlog is a list of all requirements the product has to fulfill at the

end. The sprint backlog is a list of all features that have to be done in the current sprint. This list serves to monitor the progress and whether the set goals are met [36, 95].

Scrum only works if the minimum requirements for the Scrum team are met. This means that there is at least a Product Owner, a Development Team and a Scrum Master [95]. According to [36], the development team must consist of three developers who drive the development of the software product independently and self-organized. The Scrum master helps the team to be productive and solves problems around the team. This can be directly related to the team or a possible problem created by others. He/She must also make sure that everyone in the team knows what the overall goal is. The last team member is the product owner. He has to manage the product backlog, define features or prioritize features. If changes to features or to the product are suggested, the product owner must decide whether they will be implemented or not [36].

There are also Scrum events. These are important for the process itself, as they allow the progress and possible adaptations of the process to be monitored and changed if necessary. Sprint planning is done at the beginning of a sprint. It is decided which features and which goal will be pursued. The daily meetings can be done differently, for example in the form of a stand-up. They mainly serve to measure the progress within a sprint and to communicate possible problems to the Scrum master. In the sprint review and retrospective the last sprint, problems and remaining work are discussed. The review also includes the stakeholders to capture their input [36, 95].

Kanban

Kanban is another agile process described here. It is a process model of lean software development and has been increasingly used since its introduction. There are four elements that characterize Kanban, these are pull, where work is done and not given, limited quantities, transparent information and continuous improvements. These four characteristics all pay into the value chain. In order to better represent the value chain, Kanban uses a board that represents the required phases of a company. These can vary from company to company, but usually include ready to implement, implemented, ready to test, tested and finally implemented and tested [96].

Like Scrum, Kanban also needs a self-organized team that is fully capable of doing everything in the software development process. This means that all phases must be carried out without violating set limits. These limits help the team to relieve individual persons, for example the frontend developer or the tester, because everyone in the team has to help when bottlenecks occur [96].

The meeting structure and culture is also very similar to Scrum, as it is a matter of coordinating each other on a daily basis and identifying and solving problems in advance. There are daily stand-up meetings where the progress of the project is briefly discussed [96].

For further information about Kanban, read [96, 97].

Scrum and Kanban are important building blocks for this work, as both are agile processes that can be used for software testing. In the use case scenario discussed in this paper, for example, Scrum is used, but not in the form described, and Kanban is also being used. Therefore, the focus of agile processes is on these two characteristics.

2.2 Software Testing Techniques

"Testing Can't Show That Bugs Don't Exist." R. Patton, [22]

This is an important statement and can be explained as follows: Testing doesn't proof that a software is error free, but that the tests aren't good enough to find those bugs. In [22, 21, 20] it is explained that regarding the first statement, a project manager can't think of testing as a tool that prevents errors from happening, as this is not possible. But still, software testing should be a part of the software development lifecycle as said in [21, 20]. It serves the purpose of assuring at least a level of quality for the delivered software. This depends on the time spent and the money that is put into testing the software [20]. This is shown with the resource triangle in figure 2.2. If the money is reduced and the amount of time should stay the same, the project is increased and the quality decreases. If the quality of the features should increase time and money increases as well. Testing does not directly increase the quality of the software, because the testers don't fix the bugs themselves, but by reporting bugs they enforce that they are looked at and fixed. The testers can also escalate bugs if they think that they are relevant for production as stated in [20].

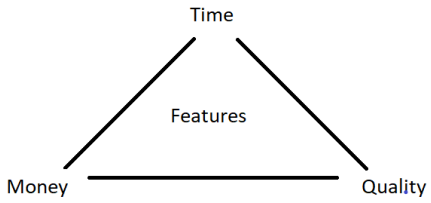

Figure 2.2: Resource triangle from [20]

It would be great to test everything, but it is not practical or sometimes even impossible to test complex systems completely and even moderate complex programs are not feasible to be tested thoroughly [20]. This is due to the costs and the time as shown in figure 2.2. This means that not every error in a system is found or could be found, because the testing would take too much time or would cost too much, but it should be done in a prioritized way [17, 16, 20].
Another important factor next to time, quality and money in software testing is risk, which is omnipresent and has to be calculated for [17, 20, 18]. Risk can be defined as:

"Risk – a factor that could result in future negative consequences, usually expressed as impact and likelihood." B. Hamling et al., [20]

If there would be no risk in developing software then there would be no testing needed, as no failures would be produced. There are two types of risk: project and product. They are both calculated as:

"Level of risk = probability of the risk occurring × impact if it did happen" B. Hamling et al., [20]

There are different types of risks for a software project such as:

- Supplier issues, if you outsource a part of a project to a third party contractor
- Organizational factors, those can include staff shortage or not enough experts on a certain topic
- Technical issues, like correct definition of requirements, meeting those requirements or that the test environment isn't set up

These are just the main types of project related risks, product risks and additional project risks can be found in [20]. The risk relates to the prioritization of the test cases as mentioned before and in [17, 20, 18]. When the risk and the probability is high, then the test has to be executed. Therefore, the risk is a direct influencer on the prioritization and also tells something about how many tests have to be done. This amount varies in every project and it depends on the completion criteria as can be seen in [20]. The prioritized tests and the completion criteria are a good measurement but the resource triangle is still the last cut. If the deadlines are not met, the quality of the software has to be comprised and the team has to decide if it's a good enough product to be released [20].

"Not All the Bugs You Find Will Be Fixed." R. Patton, [22]

This statement is interesting, as it underlines above mentioned facts. Not every bug is a bug, not everything is important enough to be fixed and sometimes there is just no money to fix the bug [22].

As said before software testing is part of the lifecycle. There are different models that can be applied to develop a software such as the waterfall model or the V-model [20].

Software testing is a discipline which splits into two main types of testing which are manual and automated, as can be seen in [11]. It is done by evaluating the behaviour of a software program with a finite set of tests. Those tests are selected because they test valuable parts of the software and are chosen carefully from the infinite possibilities of tests in the program. The evaluation which test to execute can have different influences like impact on user, generating value for the company and complexity of the feature. In [11] four key concepts for software testing are:

- Dynamic: The *System Under Test (SUT)* is executed with certain inputs to find failures in the system. This means that the SUT is tested regarding design, code and the environment.
- Finite: The test cases have to be selected from an infinite set of possible test scenarios, because in a business software there are typically various allowed, invalid or unexpected inputs and after each input the SUT has an infinite set of sequences

of operations to choose from. The testers therefore have to choose an effective test set for the available time slot.

- Selected: One of the key challenges is to select the right test cases of the huge test set that is available, that are most important for the company or find the most errors.
- Expected: The results are evaluated and categorized after each execution into failure or success.

A test run which finds no errors is exactly that and not the proof that the SUT has no errors. There is no way to tell if the complete system has no failures and even if only a sub-part is tested, a positive test run doesn't mean that there is a lack of errors as not every input value and combination of state changes can be tested. [11]

As mentioned previously in 1.2 automated testing should look something like the software pyramid 2.3 which was first introduced in [48]. This is the ideal case for every software project, as Unit testing takes far less time than service and UI testing. UI testing at the top should be as small as possible, but as stated from [48] it is an important layer nonetheless. It's just not as stable and maintainable as Unit testing. The service level is also called integration testing and should be done more than UI as it isn't always necessary to execute every test on UI level [48].

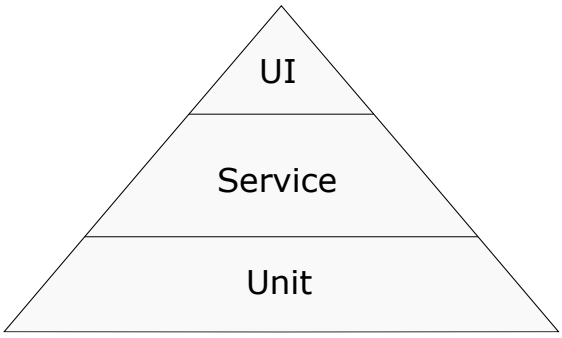

Figure 2.3: Test automation pyramid from [48]

As stated in [11] there is no overall classification of all the concepts in software testing that are available. In figure 2.4 it is shown that software testing can have various levels. The last block of the figure 2.4 are the methods which can be used for software testing.

The methods of software testing can be split in black-box, white-box and non-functional. As stated in [11] not all combinations of those previous mentioned levels are applicable, as it is not meaningful to do non-functional tests like performance tests manually or as a unit test.

Regression testing and system integration testing are other approaches to testing, but those are only briefly mentioned in [11]. Those two will be also explained in section 2.2.1.

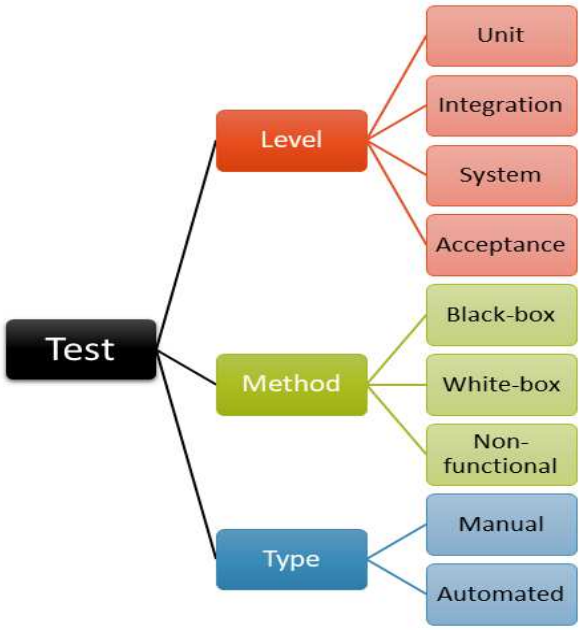

Figure 2.4: Classification of software testing into levels, methods and types from [11]

There is no universal definition of all the different testing forms and there are probably many more that will not be mentioned in this work, because it would be too much to cover.

2.2.1 Manual testing

This is the first type of software testing. The SUT is tested by a person within the company, typically from a technical background or a software engineer, or the final consumer. One specific type of manual testing is exploratory testing, where the person is investigating and/or freely evaluating the system. The person can use everything that is available and uses his/her own perception to find bugs or errors.

Another type of manual testing is regression testing [13]. These regression tests are used to find newly introduced bugs in updated software parts of the SUT. Another part of the regression tests is that they are used to test new features. The tests are executed before a release to cover all parts of the software which could possibly be harmed by the update as stated in [13]. These tests have a high demand of maintenance as they have to be updated if the flow of a program part is changed. Additionally, the execution is repetitive and boring after doing them with every iteration of the software. Even if the modified code is just a small change to the software itself, the manual testers have to ensure that the code still works in all parts of the SUT. This can lead to frustration due to doing the same thing over and over again. Another thing that has to be mentioned is that according to [12] edge cases and special test cases are not the test cases that are

executed often, but rather been left out especially by developers. But this would be the most beneficial use case for manual tests, because it is possible for the tester to find new and/or unexpected bugs. This is one of the disadvantages of automated testing, but this will be discussed in the next section.

These regression tests can often be automated, but the initial work effort is high and the maintenance of these tests is high as well. These two issues often delay the development of an automated solution for many companies as stated in [12]. Another factor are the costs of developing those tests, but in [12] it is stated that a bug can cost even more if it is not found before deploying changes of a software program to production. As mentioned in [14] 60% of the developers said that verification and validation was the first thing to be cancelled when the time and money was short and the release date came closer. This survey was conducted 15 years ago, so it could be that the value has decreased, but as can be seen in [12] there are still a lot of improvements that could be made with automating software testing.

Reducing the costs of the software, the time in which the software is built and improving the customer satisfaction is important to companies as stated in [12]. Especially hard deadlines can hurt software in the long run as it tends to be the case that developers don't have enough time to test their code thoroughly and the manual testers can not cope with the amount of work that has to be done in a very short period of time [14].

2.2.2 Automated testing

In this subsection automated testing in general is explained and how it is connected with manual testing. The focus is on automated UI testing, but the principles explained here are the same for every test stage from 2.3 but for different sizes and factors.

Every test that is done manually can be automated according to [17]. This includes tests from regression, functional, performance, concurrency, stress, and more areas. But just because it is possible to automate every manual test, it doesn't mean that it is meaningful. As it is stated in [17] one high level definition of automated software testing (AST) is:

"The application and implementation of software technology throughout the entire *Software Testing Lifecycle (STL)* with the goal of improving STL efficiencies and effectiveness." E. A. Dustin et al., [17]

The last part of the definition is important as AST should always strive to be as efficient and effective as possible and this is not always guaranteed if the software is only tested automatically. This means that you need a manual software tester as well, that enables the AST to be fully effective. Every manual tester has a certain skill-set which is important for the SUT. These skills include being analytical, test strategy know-how and an understanding of software testing techniques. Because AST is always software development, a manual tester can be used as a blueprint, which helps to improve efficiency as well [17]. As said in [32] there has to be a goal set by the management what should be

achieved by *Test Automation (TA)*. The more narrow the goal is set the better the end result is when looking at TA. Additionally the management has to agree with the TA and manage it from the top. An initial interest of one person alone is a good start, but it is not sustainable as can be seen in the many case studies done in [32]. This guidance from the management is vital to provide some quality TA and it can improve the overall quality of the product that is tested. According to [32] a deeper understanding of the matter is important as well for management to have. If they don't have the understanding it could lead to the dispense of the whole project, as they can't see the benefits gained by TA.

According to [17] AST and manual testing can't be clearly separated as they complement each other and are intertwined. As said before in 2.2.1 manual testing can be tedious, but with AST the tester is enabled to specifically test edge cases and special things, which are hard to automate or are too expensive to be automated. It is always a cost factor if AST is done as said in [12]. This is also underlined by [17], as the authors say that the *Return On Investment (ROI)* always has to be considered when automating tests, that were previously executed manually. But not just the ROI is important for the decision what to automate and what to keep manually tested.

In [32] the authors state a common mistake, that is often made by a company, which is, that only a tool that has to be acquired is the sole investment. This could mean that an open source tool is found and no investment at all is made, but this can be a bad choice, as the outcome could provide trouble, when no expert is available for this tool and bugs are not reported correctly. An investment is needed to research and experiment with different tools that are available, especially if the company wants a specific solution for their own problem. As said before the ROI is often estimated with the costs that are saved when the tests are executed manually, but it is not the sole purpose of TA and not the only way how tests cost money. As said in [32] maintenance, analysing failed tests or other tasks like documenting the tests cost money and time.

Code coverage is another factor that can be used to calculate the usefulness of TA. Others are shorter time to market and more confidence in the product, but those factors are just benefits after the TA is set up and can't be taken into consideration right at the beginning [32]. To have a successful TA periodic checks are necessary to see what has been achieved by the TA and which parts could be further improved. This can potentially save the funding and so forth the project as [32] states.

Selecting Tests for Test Automation

Here the selection process of tests to automate is shown by going through some points, that are a possible way. It is just an overview and for more details the references can be read.

According to [17] a checklist for the tests that should be automated, should be conducted as it helps to get a grip if the automation makes sense.

The table that is shown in [17] is not a complete list, as every list has to be adjusted to the situation and the company, but it gives an insight on the tests that should be

automated and a better solution to selecting these tests than just a random pick. In [17] they state that a test where every question is answered with yes, is a good candidate.

But how many of these candidates should be used for TA?

This question is answered in [17, 18]. In [18] the basics of TA are considered, like what can be automated as well as how much TA is enough. Test automation includes unit, integration and system tests followed by subcategories that include security, configuration and load tests as described in [18]. These tests should then be used for all minor and major production releases and automated regression testing should cover all of the above mentioned test fields. Additionally, the question arises when to plan for automated tests.

In [17] the automation detail is heavily weighed, like what can be automated, how much should and can it cost, is it feasible with the planned personal. Additionally, the risk has to be calculated for it and also the manual work load of repetitive tasks. These factors are still valid for today, as they are general advice. One example that is mentioned in [17] is that the test team wanted to automate every single test requirement, which was too big of a task, because the tool they tried wasn't that easy to use and optimal for the situation.

Prioritization has to be enforced with all features, as it is necessary to know, what can be done. This can be done by applying a test plan or a road map, where the list of bigger steps are summarized and then break them down into work pieces that are feasible [17]. For an in-house project the tester doesn't have a customer to serve, but it is necessary to present the results to the teams, because the motivation of the team members could decrease, if they have no feedback on what they are doing [19]. If there is a customer that the test team can report to, it is a good solution to do a repetitive meet up, to show the process and get feedback and suggestions for the ongoing work. This goes on until the customer is satisfied with the solution as stated in [17].

One factor that is repeatedly mentioned in both [17, 18] is the cost factor. This is always an important argument for the industry, as it has limited budgets and personal for the tasks. This implies a risk for the company as well, if testing is partially skipped and not done properly, as bugs can be implemented into the production software, but this has to be considered and weighed against the impact as mentioned in [17].

The risk factor which is mentioned before can be based on probability of failure of critical path functionality, an impact or business risk and the overall complexity. The test cases should be selected after determining the risk of them [17].

Unit Tests

Unit tests are the smallest and fastest tests that are executed when testing a software project, and as the name suggests test only a unit of code [11, 52]. This can be a method or a short function, such as a simple calculation. The best case scenario is that there are lots of unit tests, which have a high code coverage percentage [11, 52].

One rule that is important for all tests, not just unit tests, is, that all tests should run independently of each other and have no dependency on other tests. [8] This is especially critical in end-to-end tests, if they depend on objects that should have been created in a test that was executed earlier [55].

Integration Tests

Those tests should expose defects not only in the interface, but also when the integrated components or modules interact, as stated by [11, 8, 52]. Unit tests should be run separately and if that's the case, the integration tests can be performed with different strategies, such as top-down, bottom-up, ad-hoc or backbone integration. The advantages and disadvantages of every approach are listed in [11]. As stated by [52] integration tests should prepare the SUT to handle interaction with other code bases and frameworks.

End-to-End (E2E) Tests

Those are also called functional tests, as they test for example the login function of a web application. This can be achieved with different approaches, where one is that the *User Interface (UI)* is used to test the correct behaviour of the website, another would be to check if the right HTTP code is sent back as a response when a HTTP request is made [8].

The amount of tests in comparison to Unit and integration tests was explained with the test automation pyramid 2.3 and this should always be considered when writing tests for an application [52], as the distribution is essential for the success of testing in a project. E2E tests are tests that are used instead of manual regression tests [24]. They provide a fast and elegant solution instead of a tedious work for manual testers, as repetitive tasks are eliminated and the manual testers can concentrate on more specialized tests, which are hard to automate [18]. In [24] the authors describe the differences between capture&replay tests versus programmatic approaches, which are further described in section 2.3 in the Selenium part.

2.2.3 Advantages/Disadvantages From End To End Tests In Comparison To Manual Testing

"Automation development requires the same discipline as software development" D. Graham et al., [32]. This statement is often forgotten by the product management, as it means that the TA engineers have to put as much thought into their tests as a software developer.

Here the advantages as well as the disadvantages of end-to-end tests versus manual tests are summarized. Some of them will be mentioned in 2.2.1 and 2.2.2. The list isn't complete, as there are probably more advantages and disadvantages than those listed.

The initial set-up is cost expensive as it takes some time to get a reasonable amount of tests that are reliable, executable and cover at least the most critical parts of the software

that is tested. [17] But after the test is executed more than 10 times, the break-even point is reached as stated in [64]. This of course is dependent on some factors, such as no maintenance is needed for the test in those 10 runs or that nothing goes wrong in those 10 runs and this is only true for Unit tests. End-to-end tests are much more expensive to set up, as much more time is needed for the development and maintenance. [17, 18]. The advantage here is that repeated execution doesn't cost more, it even decreases the costs of the test, as the high setup costs are better covered. The costs for the manual tester don't decrease as he/she has to do the same work over and over again, it stays linear for the whole process. So automating regression tests are a good way of reducing the costs for software testing [17].

As stated by [17] in 2009 the main goal of software testing is to maintain or even increase the quality of the software product of a customer. This often meets the requirement of faster and cheaper shipping, which increases the stress for manual tester. The amount of work and effort that has to be put into testing can be lowered if the tests are automated, like with Unit and integration tests. Those cheap and fast to develop tests are a good way to decrease the workload for the testers [17, 64]. This enables the tester to test special edge cases or other tests that are hard to maintain, like tests that often change. This increases the overall quality of the software product as stated by [17].

Another advantage of end-to-end tests is that the tests can be run at any time of the day. As long as there is a *Continuous Integration (CI)* server running and the tests are set up to be executed automatically, any free time slot will work. This isn't possible for manual testers, as working hour laws have to be followed and overtime is more expensive [66].

2.2.4 Testing methods

The testing methods define how a test case has to be designed. In computer science there are three methods which are responsibility based (black-box), implementation based (white box) or non-functional. A combination of black- and white-box testing is called grey-box and uses both responsibility and implementation based approaches as stated in [11].

Black-box

Synonyms for black-box testing are functional or behavioral testing. Those tests are based on requirements without knowledge of the tested software. This includes data as well as the internal program structure [11].

Some of those testing techniques for black-box testing, as stated by [11], are:

- Systematic testing: This is a testing approach where the SUT is tested exhaustively until assumptions about the testing are achieved. Instead of the next technique a test is only executed if it follows the systematic approach. [11]

- Random testing: In this testing approach well formed inputs are mutated to generate different test results. [11]
- *Graphic User Interface (GUI)* testing: These kind of tests test the specification with the GUI, this includes inputs, clicks or other events on the GUI. GUI testing is operated on a system level. [11]
- *Model-based testing (MBT)*: Here a model is created for the SUT which describes at least some aspects of the SUT. This model creates the tests which can be executed on different levels. [11]
- Smoke testing: Here the critical functionality is secured with these tests. Those test cases are the first that help the testers to approve a build for further testing. If the smoke tests fail, the build isn't accepted and has to be reworked. [11]
- Sanity testing: With this technique only the basic functionality of the SUT is ensured. They are kind of similar to smoke tests as they are executed at the beginning of testing. [11]

For this work the GUI testing is the most important testing technique, as those are end-to-end tests when executed at system level. It is event-driven and acts through messages and method calls on the underlying application code. At the unit level GUI testing is usually used at the button level.

Systematic and random testing are the complete opposite. One tests completely random inputs and method calls while the other is well structured and as it states systematic [11].

Smoke tests are there to ensure that the most critical parts of a software don't fail and is the first test to be run by testers before the build is accepted for further testing. If the smoke tests fail the build will be rejected and has to be redone [11].

White-box

This technique of testing is done with knowledge of the internal logic of the code base and checks the program-code structure and logic if it is correct. Those tests only work if the workflow of the program is known. [11] Unit testing is part of white-box testing and is the basis to ensure quality in a project [8].

A combination of black- and white-box testing is necessary to select a good test-base for the SUT.

The most significant white-box testing techniques, as stated by [11], are code coverage, fault injection and mutation testing. For further explanation read some of the following references, [53, 50, 51].

Non-functional

Non-functional testing has a wide range of possible test techniques, such as performance testing, load testing, security testing or usability testing. Others are volume, stress penetration or accessibility testing. All of those can require a considerable amount of effort as stated by [11]. Some of them are intertwined, such as security and penetration testing as the later one needs authorization, which is also done by security testing.

2.2.5 Development Processes

In this subsection development processes with the focus on developing tests first are presented.

Test-Driven Development (TDD)

TDD is a pattern for software engineering, when a new project starts, because to start with coding means that tests are written. These tests are failing, as there exists no implemented code. The implementation of features happens after tests for this certain feature exist. After implementing the features, the tests should run through and even when a feature is modified and refactored, the tests should pass [52, 56, 57]. This technique is called test-first programming, and as described in [56] forces the developer to think about what could go wrong in a feature first, before even writing a single line of code. In [57] the author enlists these points:

1. Write a test
2. Run all tests
3. Write the implementation code
4. Run all tests
5. Refactor
6. Run all tests

Those steps describe exactly what TDD is all about. Run the tests often and repeatedly and try to keep the tests green. Figure 2.5 shows exactly this procedure and shows how the iteration works. The development process does not end with the first iteration but is constantly improved to deliver the best possible result. This project setup helps to reduce the chance of a failure as stated by [56] and helps to easily identify defects when new code is introduced [57]. This concept is well suited for agile companies and it's able to accelerate the development process at a huge percentage. As this isn't used at Willhaben, but could be, this is one of the process improvements that will be tackled in this work.

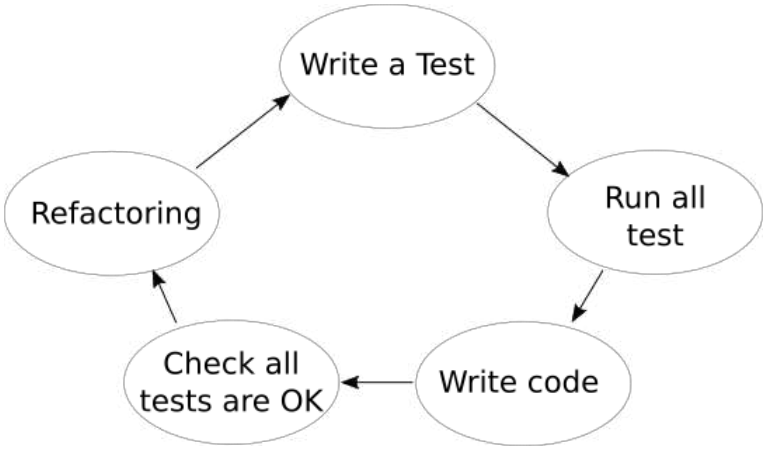

Figure 2.5: Test driven development

Behaviour Driven Development (BDD)

This technique for agile software development is based on the concept of TDD. As stated in [57] it takes TDD to the next level with writing requirements that are then used to drive and validate the developed system. Those requirements can be written in any known language and not in a coding language. In [57] the authors give some examples on how to develop a whole project with this technique and conclude that the coding principle stays like TDD, where the development process goes from failing to passing tests. In BDD all kinds of tests, for example end-to-end or integration tests, should be used [11, 57]. End-to-end tests can be written in Gherkin and Cucumber, both are explained later in 2.3.5. When describing the acceptance tests, two things happen. First automated tests are created and second a documentation is written as well [11].

When we look at BDD, it is easy to see how it aligns with iterative or agile methods, as creating the requirements up front is difficult as changes will occur [11].

Those testing methods should all be used to a certain degree in a modern project, as it is important to have unit and integration tests as those can detect bugs early in the development circle. The impact of E2E tests on the test process is one of the main factors in this work and additionally the requirements for those have to be found. TDD and BDD are both testing techniques that can be used for E2E tests and are considered for the use case, as they are a valid approach for E2E tests.

2.3 Test Frameworks

Here some of the most used frameworks will be described and explained. As said before the language that is preferred is Java, but the overall xUnit classification will be used and also Calabash which is a testing framework for many different languages will be described. The selection of frameworks is done because the huge amount of frameworks that could be used for testing is overwhelming and would break the limits of this work. The frameworks were chosen because those were the ones that are and were used by Willhaben. This does not mean that other frameworks couldn't be used as well, just that they were not the preferred ones at this one company.

2.3.1 xUnit And JUnit In Particular

One of the most used test frameworks in Java is JUnit. It is used to write Unit tests which are the fastest executable tests to test small fractions of the code base [3]. Those tests are important to detect bugs in the code and help to maintain a healthy code base in the long run. If a project has no Unit testing, or even worse no testing at all, it can end up failing, having massive software defects and never even be finished in the worst case scenario.

JUnit is the the Java implementation and was built on some of the concepts of SUnit, which is a framework for Smalltalk. Because there are many testing frameworks for many different programming languages that are based on JUnit all of these frameworks are called xUnit which refers to any member of these frameworks. They all have a certain set of features and are very similar in use and concept as stated in [5]. Most of these frameworks are written in an object-oriented programming language (OOPL) which means they have a lot of features in common. Those which are not written in an OOPL still have some of the most basic common features but differ in the more specific ones as stated in [5]. There are also extensions that provide functions of the xUnit frameworks to specific domains, like Cactus or HTMLUnit. The later is used for dealing with HTML pages and simulates a web browser [5].

There are currently two JUnit frameworks in use which are JUnit 4 and JUnit 5. The later one is, as stated on their website, "the next generation of JUnit" [121]. It's the new standard and was released in 2017. It isn't just a new release of a major version but a complete new testing framework and can be used standalone [11]. The needs for this change can be read in [11], where the author explains the motivation behind JUnit5 and major drawbacks of JUnit 4. One important aspect is that JUnit 4 was released in 2006 and since then testing evolved as well as software engineering in general. Still JUnit 4 is widely used as the step towards the new JUnit 5 is big [11]. There is a section in [11] about migrating from JUnit 4 to the new one.

The basic features consist of defining the test as a test method, using assertion methods to evaluate expected results and gathering tests into one test suite which tests something in a specific area. These test suites can consist of all the tests as well if the execution of

all Unit tests is needed. Another feature that is found in all xUnit frameworks is that a run of one or more tests will result in a test report which is needed to evaluate the tests and if and how the code base works [5]. These test reports can be used by the developers to identify bugs faster than going through each test one by one. The test reports can be exported by various tools and extensions like the Maven Surefire Report Plugin.

The architecture of the xUnit test frameworks are always built the same way with minor differences in implementation. The key classes that can be used with annotations or can be implemented are TestCase, TestRunner, TestFixture, TestSuite and TestResult [5]. The TestCase is the most important class as it contains the test function and every test class is descended from TestCase. The annotation @Test is used for that now, before that the class TestCase was extended by the test class as can be seen in [5]. The @Test-annotation clarifies that the following method is a test. The newer version with the annotation can be found in [7]. A test can be run with parameters which are set in a parameterized runner, that runs sets of tests with different parameters. This is especially useful if the test case should not behave differently for other parameters and has for example the same input fields for an end-to-end tests.

The TestRunner is a class to report the results of a test run in a better and nicer way and to run one or more test cases at once. This includes different colours when tests fail or run through and a listing off all the tests that were executed [5]. This TestRunner isn't available in JUnit 5 any more, as only one could be used at the same time [11].

Next is the TestFixture, which is a widely used tool to isolate the tests, which is considered a best practice as the dependency of a test to one other is diminished, and to give every test a clean startup process, where variables are set and test data is established. Additionally the clean up process is done by the TestFixture, and not by the test itself as this can lead to serious problems. More information about this can be found in [7]. Test isolation is even more important in end-to-end tests as it helps to reduce flaky tests and establishes independent tests that are relying only on the provided test data. In [7] a test case uses the @Before and @After annotations to create test data and a clean setup. Often variables can be initialized like this as well:

```
private final LocalDateTime timePoint = LocalDateTime.now();
```

This is just an example how someone could initialize a LocalDateTime for a distinguishable title for an end-to-end test, when used for an ad on a web marketplace which would be the use case in this work.

After the test has finished the provided test data or the instance of a web browser in the case of an end-to-end tests have to be disposed. This is as said before done via the @After annotation [7].

The TestSuite provides a container where a developer can put all the tests and start them from a single class. A test suite is declared with the @Suite annotation. A suite is just a test runner which executes all test classes with the @Test annotation that are in the same class, except if the test is additionally annotated with @Ignore [8]. Furthermore, a

master test suite which executes all other test suites with the various test can be written. Another way to execute all the tests is to add the test classes to the test suite like the following example:

```
@RunWith(ParallelSuite.class)
@Suite.SuiteClasses({
        ExampleClass1.class,
        ExampleClass2.class,
        ExampleClass3.class,
        ...
})

public class TestSuite {}
```

This can lead to problems, as the developer could forget to add test classes that were newly written. As this example shows it is possible to run tests in parallel as well, this is especially helpful if the tests contain end-to-end tests which take a long time to run, as they interact with the UI [8].

The last base component of any xUnit framework is the TestResult. As explained in [5] the test result is just a summary of all the tests that were executed previously. It is a simple object that counts the test runs, test failures and errors. Those failures and errors locate the area in which the code is faulty and provides an easy to read overview of where a potential bug can lay.

One feature that is often used in JUnit and also other frameworks is a form of test selection, where the developer can classify the Unit test even further. This helps to run smaller test sets, if the developer wants to test only some parts of the code he/she added as a new feature. He/She doesn't have to execute all of the tests that are in the project, which leads to faster test execution and therefore results. In JUnit a developer can use the annotation @Category to specify the test to a certain use case. These categories can then be executed when the category is specified in the test runner [5]. The following example shows how this can be done:

```
public interface SlowTests { /* category marker */ }

public class A {
        @Test
        public void a() {
                fail();
        }

        @Category(SlowTests.class)
        @Test
        public void b() {
        }
}

@RunWith(Categories.class)
@IncludeCategory(SlowTests.class)
@SuiteClasses( { A.class }) //  Note that Categories is a
                               kind of Suite
public class SlowTestSuite {
        // Will run A.b, but not A.a
}
```

Listing 2.1: Categorization of Tests

This can also be done by adding the test cases to the suite itself and leave certain tests out of the test suite as described before. But this has to be done with caution as a test case could be forgotten and only be added to a test suite which is only executed with a certain category.

JUnit is the java implementation of the xUnit framework and is the most used, extended and discussed framework for software testing [5]. The framework is based on the Smalltalk SUnit framework, which was the first of these testing frameworks [8]. It is the base for many other implementations and many extensions are integrated when JUnit is updated as can be seen in [7]. It is an open source project which is constantly further developed and changed.

As stated in [8] a unit testing framework should follow several best practices, which are not only viable for unit testing, because they could be applied for all testing that can be done with the frameworks. Three of these rules are as described in [8]:

- "Each unit test should run independently of all other unit tests." V. C. Massol et al., [8]
- "The framework should detect and report errors test by test." V. C. Massol et al., [8]
- "It should be easy to define which unit tests will run." V. C. Massol et al., [8]

If these rules are applied, especially the first one, it makes it much easier for the developer to develop good and non flaky test cases. This also applies for end-to-end tests, there it is even more important, because you can have inconsistent data sets, if the tests depend on each other.

These three rules are solved by or made more easily to solve by the JUnit framework as stated in [8]. The framework provides the annotations to classify the tests, a test suite where the developer can define which tests are run together and all the test results are, if the tests are executed, easily identified. The error reporting is well defined as well and as explained before it can be even be improved with various reporting tools, that work well together with JUnit.

In [8] it is stated that one of the goals that was defined by the JUnit team, was that the test costs should be reduced by the framework by reusing the already written code. This is important for this work, as the cost factor of writing tests always have to be evaluated and be compared to the actual cost, if the test is done manually.

2.3.2 TestNG

Another popular testing framework is TestNG that is based on JUnit with additional features as stated in [6]. It is one of the most used test frameworks among Java developers and does incorporate more annotations than the JUnit framework. The comparison done

here is done for JUnit4 and not 5 as the newer version was completely separated from the previous one.

These annotations are:

@BeforeTest or @AfterTest	The annotated methods will be executed before and after each test section declared inside a TestNG suite.
@BeforeSuite or @AfterSuite	The annotated method will be executed before and after any tests declared inside a TestNG suite.
@BeforeGroups @AfterGroups	These annotations are associated with the groups feature in TestNG. BeforeGroups annotated method will run before any of the test method of the specified group is executed. AfterGroups annotated method will run after any of the test method of the specified group gets executed. For this method to be executed, the user has to mention the list of groups this method belongs to using groups attribute with the said annotation. You can specify more than multiple groups if required.

Table 2.1: Annotations in TestNG. This table is from [6]

These annotations can be helpful, especially because the developer has more possibilities to set up test preconditions, that have to be executed for example before the test cases are started. There are more than those annotations but they are similar to the ones that are used in JUnit, sometimes the syntax is a little bit different [6].

The "BeforeTest" annotation lets the developer build preconditions for the tests and those are executed before every single test [6]. This can help for example to set up a test environment, in the case of TA it could be the set up for the browser or the website the test should visit.

The "BeforeSuite or AfterSuite" can be used to develop a set of usable data, that is used over all tests and afterwards clean up left behind data structures. This is especially helpful in TA when browsers crash and the connection to them are lost [6].

Another small distinction between TestNG and JUnit is the form of the assertions. The assertions in TestNG have the message as the last parameter and in JUnit it is the first parameter. This is noticeable when switching between both frameworks, but it's just a minor difference [6].

There are other small examples regarding the differences in JUnit and TestNG, those can be found at [120]. There some programming examples are given on how to write certain entities like ignoring tests or parametrized tests.

Since TestNG resembles JUnit in many different ways, as both use classes for their tests and annotations to let the developer control the set up, it isn't easy to distinguish between the two frameworks and it comes down to personal preference and knowledge which framework to use for different projects [57, 6].

2.3.3 GUI-Test Frameworks

In this part TA frameworks that can be used for End-To-End tests are described and explained. Those TA frameworks are written in different languages and are just examples of what can be used for those kind of tests, as there are many more, that are not mentioned here. The frameworks presented here are commonly known and should give a reference to other languages or as in the case of Selenium can be used in many different ones. All of the TA frameworks need an underlying test framework, which were mentioned before.

Selenium

Selenium is a test framework for end-to-end tests that is written mainly in JavaScript and C++ [10], but a developer can choose his/her favorite programming language and write the tests in the preferred language. This could be Java, but also Python, C or something else. It is working great together with JUnit or TestNG. Those are the de-facto standards for writing end-to-end tests, but if the developer wants to use another testing framework this is possible with Selenium. As described in [10] the web driver which executes the browser from the outside can interact with nearly every web browser that is available on the market. It uses the accessibility API, which is also used by various accessing and controlling applications, to drive the browser. The web driver always uses the most appropriate way to access and control the API, like for example for Firefox it uses JavaScript and in Internet Explorer C++. This is of course beneficial as the API uses the best way to access the web browser, but the downside is that new browsers or new versions with different APIs are not supported right from the start [10]. Right now the development of the tools to use the newest web-browsers with Selenium is done by the companies of those browsers as can be found here [115, 116]. They try to stay up-to-date with the browsers if changes are made in the browser itself. This wasn't the case before when the development of the API was done by Selenium itself [10] [3]. Because of that an upgrade to a browser couldn't be done until the team from Selenium had upgraded the version of Selenium and supported the new update [4]. The new way to let the browser companies upgrade the API so that Selenium can interact even with beta versions is an upgrade for all test automation engineers, as they don't have to wait to update their browsers [115]. This improves the experience of websites for the users as well, because if a user always uses the newest versions, the SUT has that covered as well. For every single browser there are new bridges as they are called from [116], which help to keep the communication stable. This includes as said before Firefox, Chrome and IE, Edge, Safari and others. Those others can all be found on the Selenium homepage [5].

In [10] the authors write about the Selenium *Integrated Development Environment (IDE)*, which is an extension for Chrome and Firefox. The Selenium IDE was not available for some time as the development team behind it, had troubles with the extension changes from Firefox as can be read here [6]. As of right now the new extension is available in the stores and can be used to capture and replay tests and locate elements that are needed for manually written tests, such a captured test case can be seen in figure 2.6. To locate an element, the user has to record his/her action via the record button in the IDE. After

that the locator that was used by the IDE can be copied and used in a manually written test [7]. The IDE and a sample of seven steps can be seen in 2.6. Every step that was recorded has a locator and the command that was executed. This tool is very effective for manual testers that have no experience with software development, as they don't need to write a single line of code [10] . A complete regression test or even a whole suite can be recorded with the IDE and can be saved afterwards. This suite, or project how it is called in the IDE, can be loaded and executed after every change that is done to the SUT. This reduces the amount of time that the tester has to put into regression testing by a lot [8,9]. The tests can be named and shared between testers. The test execution can be modified so that the tests can be run easily on different test environments. All tests that are executed can be examined afterwards and the IDE offers a simple but good enough test report [46].

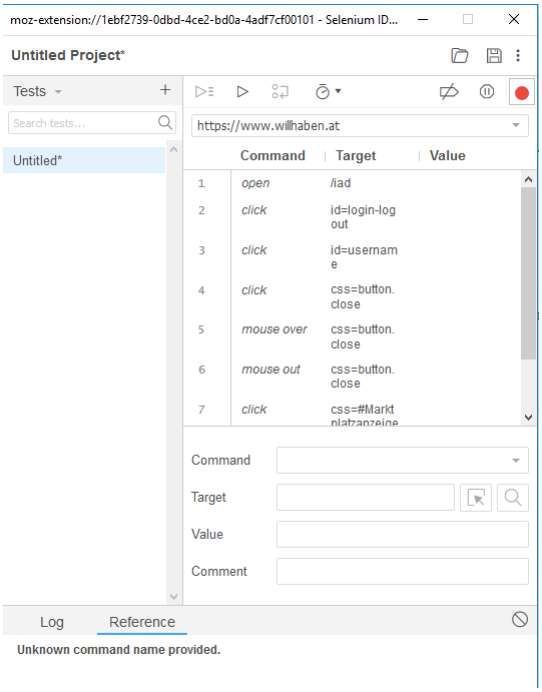

Figure 2.6: Selenium IDE

But there are limitations to the tool as well. If the test suite gets larger, the time executing them grows quickly as well, as the suite can only be executed sequentially. This limitation doesn't occur when developing the tests manually [10]. Another one is that the locators can sometimes be unspecific when using Xpath as the locating strategy. Those locators can change with every change that is done in the code, as they aren't explicit

[9]Selenium IDE, software testing;www.guru99.com/introduction-selenuim-ide.html Accessed: 2018-08-31

[9]Selenium framework, test automation; www.seleniumhq.org/ Accessed: 2018-08-26

like ids. This can drastically reduce the maintainability of the test suites, as the tests easily break. Therefore the better way to automate regression tests is to automate them manually by developing the tests and not by generating them with the IDE [10].

If the tester is proficient in software development or the TA is done by the developers it is often better to set the tests up manually. In the legacy version of the IDE it was possible to export the tests as a Java-file so that the tester could change the code if a wait had to be explicit [10]. This isn't possible any more, as this feature isn't included in the new IDE and this is good as explained next. As described in [45] a poorly written test can have various reasons. One of those reasons is that no programming pattern is used and the test is just written down in one single class. This happened when the test was exported with the legacy IDE, as the test had a poor format and was hard to read as said in [119]. An example for this is shown in 2.2. The test is simple and therefore short, but if more behaviour has to be tested, this file would become unreadable as stated in [45]. Another anti pattern as stated from [45] is the Spaghetti pattern. This isn't quite the same when talking about Spaghetti code, but it is close to it. This pattern is very fast to set up as the tester/developer doesn't have to plan, but rather just has to start coding. This leads to poor test design as the tests are interdependent and rely on each other [45]. So the Spaghetti pattern shouldn't be used for testing a bigger project and even in small ones the maintainability is horrible and maintenance is a big factor in AST [45].

```
public class LoginTest {
        private WebDriver driver = new FirefoxDriver();
        private WebDriverWait webDriverWait =
                new WebDriverWait(driver,5);

@Test
public void testLogin(){
        driver.navigate().to("www.wilhaben.at");
        webDriverWait.until(ExpectedConditions
                .visibilityOf(driver)
                        .findElement(By.id("wh-login-btn"))));
        driver.findElement((By.id("username-frontpage")))
                .sendKeys("");
        driver.findElement(By.id("password-frontpage"))
                .sendKeys("");
        driver.findElement(By.id("wh-login-btn")).click();
        webDriverWait.until(ExpectedConditions
                .visibilityOf(driver
                    .findElement(By.id("statusMessageBox"))));
        assertThat(driver
             .findElement(By.id("statusMessageBox"))
                .getText(),
        containsString("Fehler beim Login"));
}
```

Listing 2.2: Test in a single file

For this reason there is a pattern which should be used or at least something similar and this is the Page Object Pattern [46, 45]. This programming pattern is designed around objects, more specifically the page object. *Object Oriented Programming (OOP)* has been used for quite some while now as stated in [45] and Selenium web driver is written using OOP, so OOP should be a familiar term. Those page objects are a representation of web pages that have a *Domain Specific Language (DSL)* and a test can interact with the page

object but not with the actual page any more when the page object pattern is used [45]. One advantage of this is that a developer/tester doesn't have to repeat himself/herself every time he/she clicks an element. *Don't Repeat Yourself (DRY)* with the page object pattern prevents duplication of code and the implementation of various methods unlike BDD, where a single action can be phrased differently for multiple occasions of this action. Another advantage is clean code, as the test is significantly shorter when the logic behind a step is done in a page object [45]. But there are also some disadvantages such as increase of complexity, because a framework has to be developed, the tester/developer has to keep the code clean and use software design patterns, otherwise the code becomes complex to use and maintain quickly [46, 45].

Before beginning to write tests the tester has to think about how many test-cases there are and if a page object pattern is even necessary. If not, then the time should be spent on something more useful as stated in [45], as it is time consuming to set up a proper page object framework.

An example how a test with the page object pattern could look like, can be seen in 2.3. The test can be easily read because all access to the actual elements is moved from the test to the page object. In HomePage the locators for the actual DOM-elements are written and the clicks and waits for the elements are done. This would be very complex to read if everything would be put into the test and a tester wouldn't be able to easily identify the purpose of the test.

Even when the elements are found with annotations, accessing those elements can cluster up the test as stated in [46].

Selenium is a really good framework to start test automation in a company and it can be easily reused when further improvements are done, like introducing a new BDD framework to sit on top of Selenium [46]. One of those frameworks is Cucumber with Gherkin which is described in 2.3.5.

Another big factor why Selenium is often used when test automation for the UI is done, is that it's possible to use Docker with it [39]. Setting up nodes for different browser types on a Selenium Grid, which is a server where different browsers are running, is easy as it just has to be set up as explained in [39]. The Selenium Grid has an interface to check which of the browsers can be accessed and which are currently in use. Such a setup is useful, when a CI is deployed and the tests should run on this system [39]. A parallelization of the tests helps to reduce the execution time of the tests and therefore a Selenium Grid is needed as well. This parallelization can be done with JUnit or any other testing framework that supports it [7, 6]. A limitation with Docker is that it isn't possible to start an Internet Explorer as of now. This can be done manually but needs more time compared to Docker [39].

```
public class LoginTest {
        @Test
        public void testInvalidLoginCredentials() {
                HomePage homePage = HomePage.open(rule);
                homePage = homePage.loginUserExpectingError(
                    USERNAME, FALSE_PASSWORD);
                assertThat(homePage.getStatusMessage(),
                        containsString("Fehler beim Login"));
        }
}

public class HomePage extends AbstractPage {
        @FindBy(id = "username-frontpage")
        private WebElement usernameTF;
        @FindBy(id = "password-frontpage")
        private WebElement passwordTF;
        @FindBy(id = "wh-login-btn")
        private WebElement loginButton;
        @FindBy(id = "statusMessageBox")
        private WebElement statusMessageElement;

        public HomePage(SeleniumProvider seleniumProvider) {
                super(seleniumProvider);
        }

        public static HomePage open(
                SeleniumProvider seleniumProvider) {
                seleniumProvider.getWebDriver()
                        .get(getBaseUrl().toString());
                return new HomePage(seleniumProvider);
        }

        public HomePage loginUser(String username
                , String passwordText) {
                enterLoginCredentials(username
                        , passwordText);
                loginButton.click();
                return this;
        }

        private void enterLoginCredentials(
                String username,String password){
                usernameTF.sendKeys(username);
                passwordTF.sendKeys(password);
        }

        public String getStatusMessage() {
                require(statusMessageElement).visible();
                return statusMessageElement.getText();
        }
}
```

Listing 2.3: Test With PageObject Pattern

Selenium is a very flexible testing framework, as can be seen with the many use cases and configuration options mentioned above, and this is another reason why it is used so frequently [44]. There are many examples on how to use Selenium and more information can always be found at the official homepage [114].

2.3.4 Appium

Appium is a TA framework for mobile apps, such as Android and iOs, as described in [23]. Appium is written in NodeJS and could be described as a web server. The following four steps are always executed if an action is executed:

- "Receives connection from client and initiates a session"
- "Listens for commands issued"
- "Executes those commands"
- "Returns the command execution status" N. Verma, [23]

Before a test can be executed the Appium server has to be started and the server is a new instance every time the tests are executed [23]. In figure 2.7 the architecture of Appium can be seen.

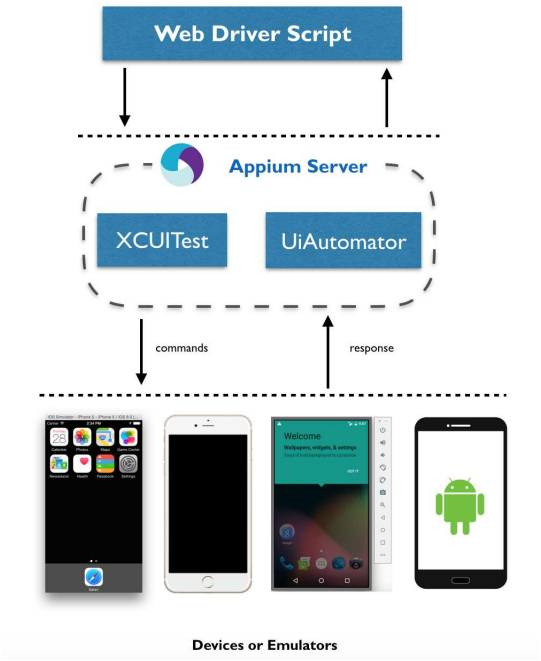

Figure 2.7: Appium architecture from [23]

XCUITest and UIAutomator2 are the automation frameworks for iOs and Android respectively. Those two are needed to build and run automated tests on real devices or emulators/simulators. Both of the frameworks install the app or apk file on the connected device, if the right configuration for it is given as described in [23].

Advantages of Appium

It is an open source tool with an active community behind it, it doesn't restrict to one specific language, it is possible to use the application that is currently available in the app store, as well as pre-compiled and built ones, it can be used for both Android and iOs and it has the Appium Inspector.

Those advantages are great, especially the active community is really worth it [23]. A second advantage is the Appium Inspector, because this tool enables the developer/tester to identify the id's, the xpath or the name of the element that is needed in the test. It is in development and gets fixes and updates as soon as a new Appium version is released [23]. If the QA team isn't that proficient with developing testcases, it is possible to record the actions, which then creates boiler plate code to help the tester as described in [23].

The Appium framework provides different drivers to use for TA. The default driver is the Appium driver as stated in [23], but there are others as well, such as the RemoteWebDriver, the Android and iOs drivers. The Appium driver inherits the RemoteWebDriver and adds additional features that are useful to use for the mobile TA. The Android and iOs drivers inherit both from Appium driver and have additional features for testing a specific Android/iOs application. The last two are best if the developer wants to test a native app, as those add helpful features that can't be used with the other two [23].

The usage of a CI system such as Jenkins should be considered, especially if the TA is an important step in the release cycle [23].

2.3.5 Calabash

Calabash is a testing framework for iOS and Android [106], but is separated into two different projects which are Calabash iOS and Calabash Android. For simplicity Calabash will always be used for both, if there are any differences between the two the specific framework will be named, so that no ambiguity will arise.

It is a framework which is developed and maintained by Xamarin, as stated in [104, 105] and can be used for native apps, as well as hybrid applications. It is an open source project with an active development, and Xamarin provides commercial services around the framework like the Xamarin Test Cloud where the Calabash tests can be executed on a large variety of Android or iOS devices [104, 105]. It is a conglomeration of different libraries, which enable interaction with the apps. Those interactions can be gestures, assertions or screenshots [106]. Assertions are written the same way as when the developer writes the code for a Unit test in the respective language.

The tests for the framework can be written in Ruby as well as other popular languages like Java, Swift or Objective-C [104, 105]. Ruby is easy to pick up, if the developer or tester has skills in software engineering or has learned another object-oriented programming (OOP)-language already [29]. Ruby is a small, intuitive language which consists mostly of sensible single words. It has no compilation phase and therefore gives the developer an immediate response on what he/she writes. So if the tester is eager to learn a new programming language he/she can do that by picking up Ruby, while providing TA for the applications [105, 29, 104].

Calabash is used in combination with Cucumber, which enables non software developers to write those tests, because many steps are predefined and can just be executed by the script [107]. Cucumber is a framework that is used for behaviour-driven development

(BDD). It is written with Gherkin which is a domain-specific language, that is specialized in describing features and scenarios. [30, 31]. Gherkin is developed so that the language is very similar to a natural language and therefore is easily understood when read by someone with a not technical background. [30]. In [31] the author explains how to use this powerful tool to create such easy to read scenarios and after that how to write the underlying code for those tests, as the scenarios are the tests that are executed.

Another effect that Gherkin often enables is that the specifications for features can be written beforehand in the scenario and the test that is underneath the scenario provides information for the developers as well. So a domain expert can write those specifications without knowing anything about programming. And if this isn't possible, it is possible to just read the feature afterwards, if it was written by somebody else, as it is near the natural language. The only exception are the keywords that have to be written down as described in [31]. A sample feature file can be seen at code block 2.4. It is a very simple feature file, but it shows what can be done with the feature. In [31] it is mentioned that the feature tag should consist of business requirements and can contain an intent text about the feature.

```
Feature: We want to test the login on the app

Scenario: Login to the app

Given user is on homepage
When user logs in with valid credentials
Then user is logged in
And own ads are visible
```

Listing 2.4: Example for Cucumber feature file.

As described in [30], BDD aims to improve the communication between domain experts and software developers. This can be achieved by letting the domain experts write the feature files themselves. Especially the description of the feature and the scenario can provide vital information and can be written in any language the author wants [30].

In [31] it is explained how complex data types can be used in the Gherkin feature file, as the writer of the file can provide the test data beforehand and also use predefined "variables". These variables have to be passed and interpreted in the step definition, which is the code construct behind the feature file. Those arguments are regular expressions and have to be written like in 2.5. When some part of the code is surrounded by parentheses, this means that those are part of a regular expression and become a capture group. This text is then passed to the code block as an argument.

```
Feature: We want to test the login on the app

Scenario: Login to the app

Given user is on homepage
When user logs in with email "test@test.at", password "1234"
Then user is logged in
And own ads are visible
```

Listing 2.5: Example for Cucumber feature file with variable.

Such a step definition can be auto generated by most IDE, and in [30] the author describes how the execution of the feature file, without any other code, creates a report, that can be used by the developer to see what step definitions are missing and what to code next. If the steps aren't created the test report would state that they are undefined [31].

An example step definition can be seen in 2.6. Running the test after creating the step definitions would now state that one step definition was pending and the others were skipped [31]. This is an indication that the step definitions have been created but no code was implemented for them.

```
package com.StepDefinitions;

import cucumber.api.PendingException;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class LoginSteps {

        @Given("^user is on homepage$")
        public void user_is_on_homepage() throws Throwable {
                /*Write code here that turns the phrase above
                into concrete actions */
                throw new PendingException();
        }

        @When("^user logs in with valid credentials$")
        public void user_logs_in_with_valid_credentials()
                throws Throwable {
                /* Write code here that turns the phrase above
                into concrete actions */
                throw new PendingException();
        }

        @Then("^user is logged in$")
        public void user_is_logged_in() throws Throwable {
                /* Write code here that turns the phrase above
                into concrete actions */
                throw new PendingException();
        }

        @Then("^own ads are visible$")
        public void own_ads_are_visible() throws Throwable {
                /* Write code here that turns the phrase above
                into concrete actions */
                throw new PendingException();
        }
}
```

Listing 2.6: Example for a step definition.

As stated in [31] the test report indicates the successful implementation of code as well, as the pending would be replaced by passed or error. This is very useful for the developer as he/she can tell immediately if something is missing in the implementation. The usage of regular expressions for the variables that were described earlier can be found in both [30, 31].

The tests can be executed on the simulator, the emulator or the real device respectively. The emulator can be slow as those emulated devices consume a lot of computing power and sometimes don't behave like real devices [108]. Real devices are expensive to be purchased but this will be explained later.

In subsection 2.3.6 the difference between emulators, simulators and real devices are explained and compared.

Calabash is easily configured to run tests on many different devices [107]. This provides a lot of power regarding different form factors, OS versions and hardware specifications. Calabash in itself is considered to be a BDD framework, as it describes the application behaviour and not the shape of the API [107]. Calabash has built-in steps and methods that are tailored for mobile applications to simplify coding tests and make it a fun experience [107].

Another important feature of Calabash is that it can be integrated by a CI- or an continuous delivery- system such as Jenkins [107]. This can be done with many frameworks that were listed here as mentioned before, but it is still a plus point for Calabash.

2.3.6 Emulators, Simulators or Real Devices

There are a lot of options to choose from, when it comes to test execution with a mobile application. Emulators and simulators are often provided for free, such as from GenyMotion or XCode. There is a native version of an Android simulator as well, if the Android Studio IDE is used, where a user can download all sorts of different sdk versions, to execute the tests on. Now to the comparison between emulators, simulators and the real devices.

Disadvantages of emulators

- "Mobile device emulators are very slow (because they simulate both hardware and software)"
- "A mobile device emulator doesn't take into consideration factors like battery overheating/drainage or conflicts with other (default) apps"
- "Setting up a good emulator takes time and it is expensive"
- "They may be incompatible with the app or app elements, meaning that you will need to create patches here and there to keep on using the emulator"
- "Emulators may support only certain OS versions" [108]

Advantages of emulators

- "Simulates both software and hardware" [108]
- App runs on hardware as well
- Can detect unexpected behaviour
- Normally open source

- Inexpensive

On the other hand it is possible to run the tests on a simulator, which has advantages and disadvantages as well over the emulator. For instance it is fast, as it only simulates the software [108]. This can lead to problems as the hardware isn't considered at all and the app therefore could run differently than on an emulator or a real device. A simulator can be set up very easily as stated in [108].

Those two options to run tests on a device are suitable and can help bring the TA on an app to life, but it should always be the plan to test the application on a real device as every emulator or simulator has its problems. [108]. In the early stages it is a good plan to use emulators/simulators as setting up a device farm is expensive and there are always new phones coming out and a company, that isn't specialized in providing a device farm, can't keep up with the constantly growing market of smart phones.

Real devices have additional disadvantages, besides the fact that they are expensive. The devices have to be maintained and not every model of a phone is available in every market around the globe [108]. But those disadvantages are outweighed by the advantages, except the cost factor.

Here are just a few advantages from real devices as stated by [108]. One is that testing is done in a real environment. Others are that push notifications can be activated and tested, testing can be done on a real device, with the android/iOs version of the device, bugs that are reported by users can be replicated easily and performance or lack of it can be tested.

As said in [108] there is a fourth option to this problem, as it is possible to use a cloud service, such as Saucelabs or BrowserStack which both offer solutions for the expensive upkeep of an own device farm.

2.3.7 Summary

In the previous sections software testing and various variations of it were discussed as well as frameworks that could be used for the purpose of test automation and E2E tests. There are quite a few options to select from and because of that it's not always easy to find the best possible solution for a specific company. As so many options are available it would be best to have best practices listed on which frameworks should be used for which case, which tests should be used for which purpose and how everything can be organized. This is done in this work with analyzing the current situation at Willhaben as a single use case and this provides a sample which best practices should be used.

The E2E tests and their requirements are focused on this work as well and with the different possibilities on how to write tests, as well as how much should be tested and if there are use cases where another approach should be used. As shown in the previous sections there are a lot of requirements that have to be defined and it differs for every company that is evaluated in this process. Those requirements are identified as an example at Willhaben and other classifieds, which are connected to Willhaben via Schibsted.

CHAPTER 3

Software Process Improvement with Process Lines

Software processes are related activities which lead to the production of a software. This can either be for an existing project that is extended or changed or a completely new software. There are different approaches to improve a software process. Some of them will be listed here to get an overview on how such an approach could look like. As stated by [71] the software process improvement itself is a process improvement as well. This means that if a software process is changed, the process can't be isolated from other processes that are involved for the development of a software. As [71] says most of the techniques and methods that are used for any other complex people-centred activity are required for the improvement of a software development process. This is supported by [73] as they state that they found three critical dimensions which companies focus on and those are people, procedures and methods, and tools and equipment. "A software process model defines the standard process that provides the basis for organization's process assessment and improvement" S. Peldzius et al., [80]. This defines what a process model is and how it should be understood and in the following sections different process models are explained.

3.0.1 Plan-Do-Check-Act (PDCA)

Plan-Do-Check-Act is a process improvement approach that cycles through the different stages of plan, do, check and act. This can be seen in figure 3.1 which is from [78] and illustrates the different sections of the process improvement. As the cycle suggests, it is an endless improvement as the loop has to start again after finishing the previous one. In [77] the author states that the ISO 9001 process also uses the PDCA cycle as a tool for process improvement and as can be seen in figure 3.1 the ISO certification is a bullet point of the stage act.

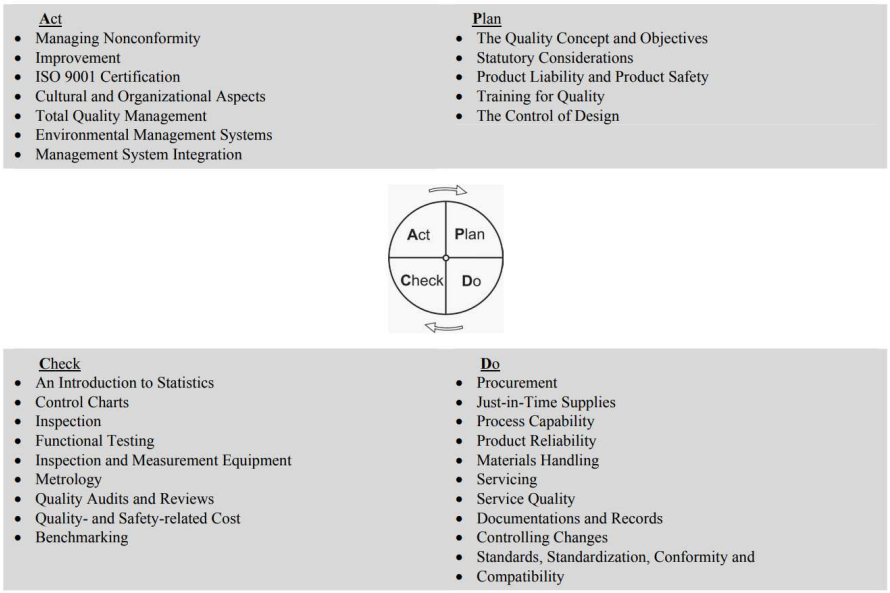

Figure 3.1: PDCA Cycle from [78]

The PDCA cycle is also regarded as Deming circle after the founder. A similar approach is the PDCA which is plan, do, study and act. As can be seen in figure 3.1 there are many focus points for the process to be improved. As [78] states the PDCA is used to find better methods continuously as in comparison to "the right first time" approach.

PDCA has a very general approach on process improvement and isn't specifically for software processes, as it could be applied on every process in every company. As it doesn't have any specific software process areas it isn't the right improvement approach for this work. The general concept of improving the process continuously is a well designed approach, but this is also done in the other process improvement approaches that are mentioned as well.

3.0.2 ISO 9001

This is a standard for quality and process management and has a common concern with the *Capability Maturity Model Integration (CMMI)* which is explained later [76]. The ISO 9001 standard is needed when a supplier of a certain product is required to demonstrate its ability to design and supply this specific product to a second party. Those parties can be partners where one is an external client or it is also possible that both are in the same company, but in different departments. ISO 9001 is the most relevant standard for software development and software quality. It is used when a company must ensure if it meets the required specifications during a period of time or stages of the software development cycle. The ISO 9000-3 standard is supporting the ISO 9001 in terms of guidelines for the development, supply and maintenance of the software [75]. Auditors

that hand out certifications may not be qualified or have knowledge about software and software development, but if the scope of an audit is software related an auditor with knowledge should be included.

In [77] the changes for the ISO 9001 standard are described. One of those changes is that the organizations have to take a look into the future on how they perceive themselves in 5 or more years and where they want to be in that time. This helps them to focus on what they want to improve and which activities have to be done to reach their goals. Because of the focus of the previous standard on having a plan for the actual situation and the situation in a few months the planning was short- or medium-termed. This should change now with the ISO 9001:2015 standard, but as [77] describes there are no demands on actions that have to be done.

Another change is that employees have to be more aware about quality, process and customer orientation. This has to be directed by the leader or head of the department and he/she has to not only convey knowledge but also explain the correlation between the different topics. There should be a real plan of action so that employees can execute it and help the company to reach the given goals [77].

The plan-do-check-act principle is integrated in the standard and is required for the certification process and it is necessary for the 2015 ISO norm [77].

The ISO 9001 isn't a software development specific standard, but more a general approach on process improvement and as this work is strongly related to software engineering and software development, it's not the ideal process improvement approach. Therefore the CMMI and SPICE standards, which are more specifically targeting software process improvement, are considered as the go to approach for this work.

3.0.3 Capability Maturity Model Integration (CMMI)

CMMI is used for the development of products and services and is a process improvement maturity model as stated by [73]. It uses best practices to address the development and maintenance activities that are necessary for the product lifecycle. CMMI is used from the start of the project until the product is delivered and maintained. The CMMI was developed and evolved by the university of Carnegie Mellon, people from industry and government [73].

As said before the three critical dimensions are what is typically focused on by companies. The processes that are used in the company are the glue that keeps everything together. They aren't only aligning the business that is concluded, but additionally provide ways to address scalability and how to do things in a better way [73]. This doesn't exclude people and technology from the equation, as people provide knowledge and have to work with the processes, and technology is always changing. But with those processes the full potential of the people can be unlocked and help them to raise productivity. Also new technology can be used far quicker and therefore give the company an edge over competitors [73].

"The *Capability Maturity Model (CMM)*, including CMMI, is a simplified representation of the world" C. P. Team, [73]. The goal of CMMs is to improve the processes in an organization. The CMMI is one of many CMMs and provides guidance for developing processes but isn't actually a process or a process description. It just measures the maturity of a process and provides best practices for software engineering. [74]. Those processes are unique to every organization as they depend on many factors, such as size, application domain and the organizational structure. CMMI can't provide an out-of-the-box solution for an organization because of the mentioned limitations [73].

There are three model components that are defined for CMMI and those are as stated from [73]:

- **Required**: These components are as the name states required for achieving the process improvement in a process area. They are the specific and generic goals and if these goals are met the process area is considered satisfied.
- **Expected**: These are components that help the persons that implement or perform the process improvements and describe the activities that are needed when the required component has to be achieved. The goals that are planned to be satisfied can only be satisfied when the practices or acceptable alternatives are present in the planned and implemented process.
- **Informative**: Informative components are for example notes or detailed explanations that help users understand required or expected components. As [73] states understanding the model without informative components is nearly impossible as those are necessary to correctly understand the goals and practices of the model.

But these components are just one part of the basis for CMMI, as there are process areas as well that are defined by the team of CMMI. These areas are the main factor for the categorisation of the processes and therefore the basis for every further step. A process area, as described by [73] is a cluster of related practices in a specific area, and helps to improve this area when certain important goals are satisfied. As of 2010 [73] defined 24 process areas. The least information these process areas must contain, as listed in [74], are:

- Name and abbreviation of the process-area
- Category and grade of maturity of a process-area
- Definition and purpose
- Specific goals and practices

There are 24 process areas, of which Process and Product Quality Assurance, Risk Management, Work Monitoring and Control and Work Planning are especially interesting

Level	Continuous Representation Capability Levels	Staged Representation Maturity Levels
Level 0	Incomplete	
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4		Quantitatively Managed
Level 5		Optimizing

Table 3.1: Comparison of Capability and Maturity Levels from [73]

for this work, as they can be related to software testing. The full list can be found in [73] and as well in [74]. It states these process areas can be clustered in different categories. Those categories are development, project management, process management and support.

In CMMI there are different capability and maturity levels that can be reached when certain aspects of the process improvement are implemented and executed. The capability levels are specifically used to measure the process improvement in a certain process area. There are four different capability levels that can be achieved by improving the processes for a given process area. The maturity levels on the other hand consider all process areas that are important for the company and measures the improvements on those multiple process areas. The capability levels are labelled from 0 to 3 while the maturity levels are labelled from 1 to 5 [73].

In table 3.1 the different capability and maturity levels are displayed. There are two levels that have the same names with managed and defined, the other levels either aren't available for both or the naming is different. As stated by [73] both levels provide information about how far a process, and therefore the company, has improved and a way to get even better.

Capability Levels

As said before there are four different capability levels. Those are:

- Incomplete
  An incomplete process is a process which has one or more specific goals not satisfied and no generic goals as there is no reason to create those when the process isn't performed.

- Performed
  Here the process was performed so that work products were produced and the specific goals are, as in contradiction to level 0, satisfied, but there are still no generic goals.

- Managed
  If a process is on capability level 2, this means that there is policy for the process to be executed by someone with the knowledge and proficiency to do so, is monitored and controlled, and is evaluated afterwards. In the end this process can be done like any other task, as it has all the requirements.

- Defined
  A defined process is a managed process which is specifically fitted for a certain group of people instead of being a standardized process, which can be performed by the whole company.

Those levels are from [73]. When the last capability level is reached for certain process areas, the company can aim for improving high maturity process areas. If this is done, the processes that are already in place are being targeted to be improved even further.

Maturity Levels

These levels describe the current maturity state and the performance of a certain process area as described in [73, 74]. The process areas in which the process improvement should be done has to be selected carefully as not every process area can be handled simultaneously and so the most important ones have to be prioritized. The specific and generic goals of a process area are the measurements for defining the current maturity level. The capability and maturity levels are complementary and this is shown with the naming of two of the levels. The five maturity levels are:

- Initial
  Here the processes are chaotic and unpredictable, as the environment isn't stable enough to support processes. Characteristics of this level are that the successes often can't be repeated or that processes are abandoned if a crisis appears. Work often gets delayed and is above the budget.

- Managed
  At the managed maturity level the processes are managed on a project level. The process can be reactive and work groups, work activities, processes, work products and services are managed. The process discipline helps to execute existing practices even in the time of stress.

- Defined
  Here the process is done proactively instead of reactively as in level 2. There is a set of standard processes and those are established and are being improved over time. The process isn't only for a work group but for the whole company and the set of processes are helpful as they bring consistency into the organization. The standard processes are tailored for and by the work groups. This means that the processes are more consistent throughout the whole company.

- Quantitatively Managed
  The processes at this level are measured and controlled by quantitative analysis and statistics. They have certain objectives for quality and process performance and those are based on the needs of the customer.

- Optimizing
  In this level the company focuses on improving the processes constantly considering the business objectives and performance needs. This is done by improving the process incrementally. The main difference to level 4 is that the organizational performance and its improvement is in the focus.

As said by [76] there are key process areas which have to be achieved when going from one maturity level to another. Those will be not listed here as it can be found in [73, 76].

In comparison to the ISO 9001 standard the CMMI is a software process improvement approach, but is not as specialized as SPICE as it has no process area which is directed to software testing. This is explained in 3.0.4 in more detail.

3.0.4 SPICE or ISO/IEC 15504

Software process improvement and capability determination (SPICE) is a software process improvement approach, which is provided by the ISO [74] and is one of the two most used SPI's in the world. SPICE is more often used by government supported projects while CMMI as the second standard is often used in software companies [79].

It is based on two main focuses, the first one is capacity determination and the second is software improvement. Especially the second one is important for this work as it can be used to highlight process areas where improvements have to be done. The first cornerstone is about rating already existing processes and improving those processes with this rating. As stated by [79] customers demand more quality from software services and not just from the software products that were produced out of mature processes. SPICE uses a similar approach to rate a process that is already implemented like CMMI, which takes the maturity of a process into account and evaluates it. This is also done with SPICE, but in different processes. As said by [80], this staged representation model is used to rate the maturity of the software process. Every single maturity level provides a basis for the one above itself. The result of this rating is the maturity level. The second model is the continuous representation model, which is used to rate the capability levels of each process area [80]. ISO 15504 uses the organizational maturity framework with the introduction of ISO/IEC 15505-7:2008. It defines 6 maturity and capability levels from immature to optimizing, and incomplete to optimizing which corresponds to level 0 to 5. Each of the maturity levels includes a defined process capability profile, except for level 0 [80]. The process assessment has two indicators which are important and those are process capability and process performance [82].

Capability Level	ISO/IEC 15504 Capability level description
Level 0 Incomplete	The process is either not implemented or doesn't achieve its purpose
Level 1 Performed	The process is implemented and achieves its purpose
Level 2 Managed	The process from level 1 is implemented as a managed process, therefore its planned, monitored and adjusted. The work products from this process are established, controlled and maintained.
Level 3 Established	The previous process is now achieving its process outcomes with the help of a defined process.
Level 4 Predictable	The process outcomes are now predictable, whenever a variation arises a corrective action is taken. This done by analysing measurement data, as well as quantitative management needs.
Level 5 Optimizing	Improving the process continuously when an organizational change happens

Table 3.2: The different capability levels of ISO 15504 from [82]

Measurement Framework

This framework is responsible to define the necessary requirements and rules for every capability dimension and on which capability level a certain process is. For this purpose the framework provides process attributes [82].

In table 3.2 the capability levels of the standard can be seen. In those 6 levels there are 9 process attributes, which are:

- Level 1:

  Process performance attribute

- Level 2:

  Performance management process attribute

  Work product management process attribute

- Level 3:

  Process definition process attribute

  Process deployment process attribute

- Level 4:

  Quantitative analysis process attribute

  Quantitative control process attribute

- Level 5:

  Process innovation process attribute

  Process innovation implementation process attribute

This list of the process attributes is from [82]. A process attribute can either be not, partially, largely and fully achieved. The percentages for each level of achievement are:

- 0 to $\leqslant$ 15% for not achieved
- 15% to $\leqslant$ 50% for partially
- 50% to $\leqslant$ 85% for largely
- 85% to 100% for fully achieved

Because the levels for partially and largely achieved are quite wide, it is possible to use further refinements, which are described in [82]. Process attributes are essentially features of a process that are used to measure the capability of this process. Every process attribute is applicable to any process that is used in SPICE. Each of these process attributes is assigned to a capability level.

Process groups

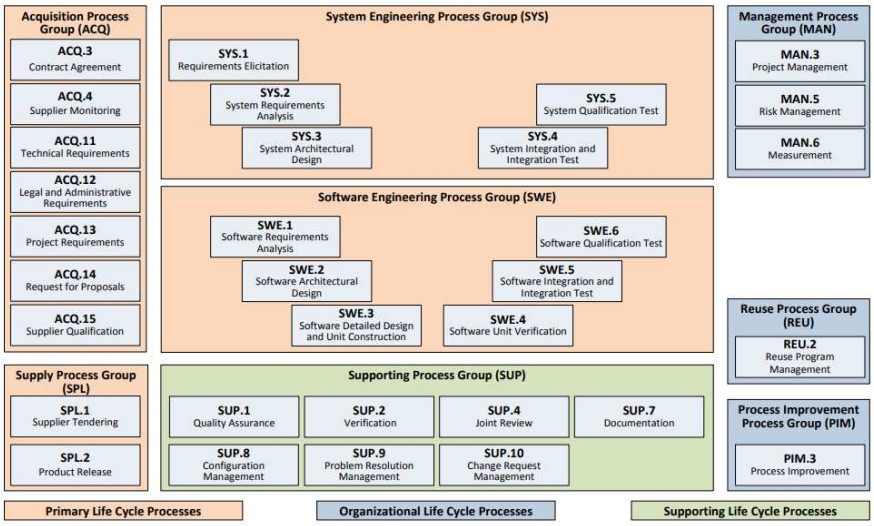

Figure 3.2: Automotive SPICE process reference model from [82]

Those processes are grouped in different process categories. Those 3 categories are primary-, organizational-and supporting life cycle processes. In figure 3.2 an overview

of every process group and process that is covered with SPICE, is shown. The most important subcategory for this work is the *software engineering process group (SWE)* as it includes all kinds of testing [82].

The SWE is part of the primary life cycle processes category, which is as the title says the centre of the focus. Another important group is the *system engineering group (SYS)*, which is also part of the primary group, which addresses integration and system testing, as well as other system related topics [82].

From the organizational life cycle processes, the project and the risk management are important as well, as testing is part of the project and has to be managed and the test selection is always done with prioritizing the tests that cover high risk areas. The process improvement which covers the process of improving the organizational processes is also worth looking into, as the processes in the Schibsted have to be looked at and improved [82].

As the last process category the supporting processes come into play, which consist of *Quality Assurance (QA)*, verification, problem resolution management and change request management as the important ones for this work, which can all be applied to other processes at any time in the life cycle [82].

The processes can be rated with three different methods. Those are:

Method R1:

"The approach to process attribute rating shall satisfy the following conditions:

(a) Each process outcome of each process within the scope of the assessment shall be characterized for each process instance, based on validated data;

(b) Each process attribute outcome of each process attribute for each process within the scope of the assessment shall be characterised for each process instance, based on validated data;

(c) Process outcome characterisations for all assessed process instances shall be aggregated to provide a process performance attribute achievement rating;

(d) Process attribute outcome characterisations for all assessed process instances shall be aggregated to provide a process attribute achievement rating." SIG-Automotive, [82]

Method R2:

"The approach to process attribute rating shall satisfy the following conditions:

(a) Each process attribute for each process within the scope of the assessment shall be characterized for each process instance, based on validated data;

(b) Process attribute characterisations for all assessed process instances shall be aggregated to provide a process attribute achievement rating." SIG-Automotive, [82]

Method R3:

"Process attribute rating across assessed process instances shall be made without aggregation." SIG-Automotive, [82]

These three process ratings rely on wether the rating is done on a process attribute level or on the process attribute and the process attribute outcome level and on the summary of ratings on the rated process instances on each process [82].

Which process capability level a process has achieved is decided by the process attribute ratings. Here only the levels 1-5 are considered, as level 0 can't achieve anything. The levels build upon each other so for level 2 level 1 has to be fully achieved, and so on, as well as the level is achieved when all process attributes of the corresponding level have been largely achieved.

Process Assessment Model

As described in [82] the process assessment model provides two different indicators with which the process outcomes and the process attributes outcomes are in place or missed. The indicators are not a checklist that has to be followed but rather provide a guidance if a capability level is reached.

These two indicators are process performance indicators and process capability indicators. The first one is only applicable on capability level 1 and indicates if the process outcomes are sufficient for this level. The process capability indicators are used in level 2 to 5. They are indicating if the process attribute achievements are sufficient [82]. The assessment indicators help to check if a certain practice was performed by examining the work products of the processes or from statements of performers and managers that are the owners of the process.

The process performance indicators consist of two types, which are *Base Practices (BP)* and *Work Products (WP)*. Those two are never generic but process-specific and are related to one or more work process' outcomes.

The types of the process capability indicators are *Generic Practice (GP)* and *Generic Resource (GR)*. As the name suggests they have to be generic and therefore apply to every process. They are related to one or more process attribute achievements.

In figure 3.3 the different GPs in each level are shown. At level 0 there is only one GP as every level requires at least one and level 0 is as stated before rated with performance indicators. This GP is an editorial reference to these indicators.

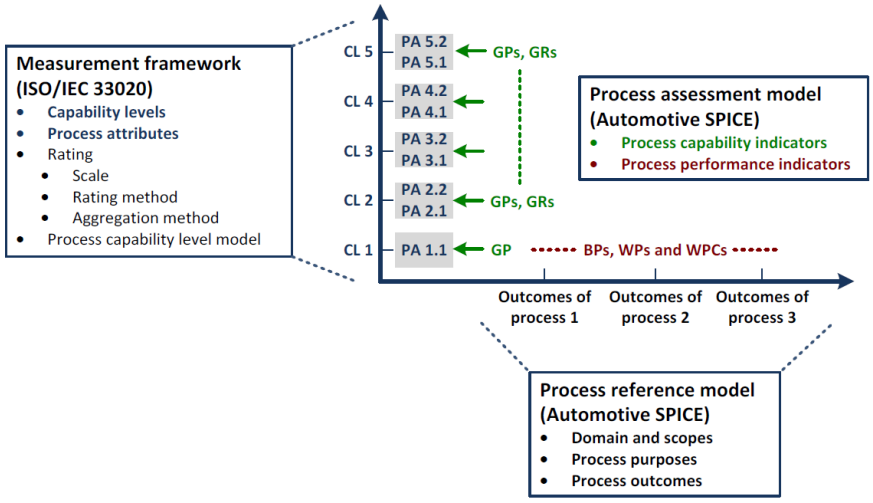

Figure 3.3: Relationship between assessment indicators and process capability from [82]

Process Reference Model

Every process is divided by its area by the *Process Reference Model (PRM)*. To classify the process the output is taken into account as described by [74]. The PRM is used to describe the purpose of each process and the usage of it. As said before there are some process areas which are more interesting for this work than others, such as the SWE. The PRM describes in [82] every process group additionally to the name and id with the process purpose, the process outcomes, the base practices that are needed to fulfil the process and the output work products.

In this work the more important process groups and the associated processes are described, for more detailed information about the mentioned processes and information about others, read in [82].

The system integration and integration test process is the first in the list and the purpose of it is to provide an integrated system, that is in line with the system architectural design. The tests are there to ensure that the system is tested and capable to be delivered. The process outcomes include a system integration strategy that is within the project plan, a release plan and the system architectural design, as well as the tests that were developed for this.

The system qualification test process is in place to check if the system is compliant to the system requirements and that it is tested. Part of the process outcomes is that there is a test strategy with regression tests in place and the project and release plan are tested with this test strategy. The tests selected are then used for testing the integrated system.

From the SWE the software qualification test process has to be mentioned as this process ensures that the integrated software is tested. Therefore, a software qualification test

strategy is needed, as well as developing the tests that are necessary to provide evidence that the software requirements are met.

Because of all these traits in SPICE regarding software testing, this software process improvement approach is chosen for this work instead of CMMI or ISO 9001.

3.1 Software Process Line

In [82] it is mentioned that creating a specific process for a team, product line or company doesn't mean that this process is completely applicable for every team, or for every product. Therefore this work uses a software process line to check if something like a product line is possible in a process environment.

3.1.1 Software Product Lines Overview

A *Software Process Line (SPrL)* is closely related to *Software Product Lines (SPLs)*, where the possibility of software reuse in a company or a corporation is explored, as it is an SPL in the process domain [65]. The definition of an SPL is:

"A software product line is a set of software applications that shares concerns, features, requirements, or market specificity and that are built in a rational and planned way from a set of re-usable assets." H. Royer et al., [70]

The use of SPLs has started, when companies discovered the resemblance between software and classic production, like car manufacturing, where many parts are reused for the different products and assembled in the same factory as stated in [62] and saw the potential of an advantage against competitors. Reusing certain parts of software or even whole software artifacts, which are used more than once across multiple functionalities is a good way to reduce the risk of problems of software development [70]. SPL engineering is one of two approaches for software engineering. The other one is model-driven engineering, which isn't discussed in this work, but can be found in [70]. SPLs have a common software architecture which is than used to do a domain analysis and scoping activity. This process characterizes the products that should be delivered [70]. SPLs main goal is to increase productivity and this has several outcomes. Saving time, reducing costs and increasing the quality of products are those three aspects, which are also important for automated software testing (AST), as mentioned earlier in 2.2.2. These three outcomes or better called concerns are mentioned in [70]. SPLs could help at AST as well, as a lot of test code can be reused. A software testing process line would be helpful here as well, as it could provide a process for software testing and a complete test process with verification and quality assurance as well.

Software which is created with the help of an SPL uses existing parts of other software artifacts to minimize redevelopment of a feature, which is essentially there but need certain tweaks and changes [70]. The used artifacts are mostly from the same domain and share concerns. To give an example how an SPL could look like, [70] mentions

Smart-Home systems, which have commonalities, such as rooms, doors and several other parts and also a set of differences, which is called variability, such as configuring a home with automatic windows or one with automatic doors. In [70] Arboleda gives a lot of examples and a history on how SPLs looked like. The main processes used for SPLE are the management process, the core assets development and products development [70]. A simple and abstract view of those processes is shown in figure 3.4.

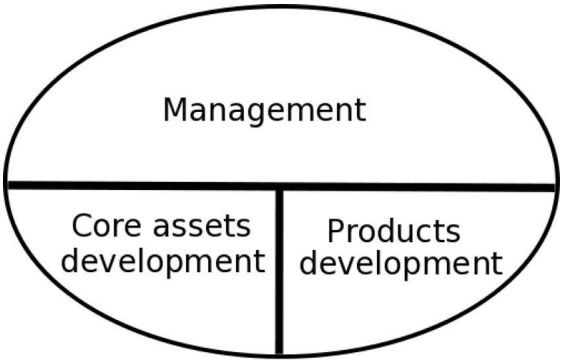

Figure 3.4: Three processes view from [70]

As can be seen, the management process serves as a supervisor above the other two. It's the process which is in control of the business plan and checks the quality of the core assets and final products. Individual elements are in the responsibility of the core development asset as well as to make those elements reusable. As stated by [70] it defines the product line scope, which describes the set of products, and the production plan, which is for building the different products. With the third process, the products development, the products, that are defined in the production plan, are built. This includes the description of the product where the features are described and then goes into the implementation of the product with the existing core assets. If additional assets are needed, they will be suggested.

Figure 3.5 shows the domain and application engineering, which are both used for SPLs. The domain engineering checks the domain, the common features and at last the variability. This means that this is used to get the core assets, which can be reused for newly developed features. The second is like the products development process responsible for the implementations. The five main activities, which are shown in figure 3.5, describe those previous mentioned steps better, as they include everything that has to be done, when using these two engineering practices.

The variability model is an often mentioned concept in [70] and [62, 63]. As said before variants of different aspects of the system together with the commonalities are the key driver for an SPL and define the product line platform. In [70] the authors state that there is no standard definition of how to represent variation points and variants in a variability model. The separation between software-design concerns and technological platform concerns is desirable, according to [70], as this was the trend in 2013. This

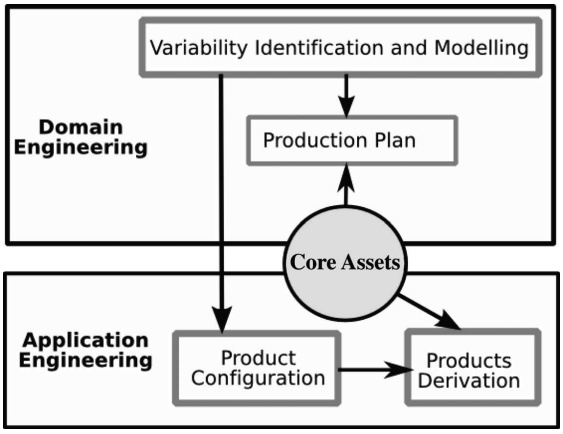

Figure 3.5: Domain and application engineering from [70]

helps the architects of an SPL to concentrate on the different concerns at different times. Feature models are the de-facto standard for modeling variability and for showing the relationship between different features with a tree-like structure [98]. In [99] de Carvalho et al. state that variabilities in an SPrL are represented by points of variation and optionalities. Those variation points represent process elements that can take different forms.

3.1.2 Software Process Lines

Now let's have a closer look at what an SPrL really is, how it can help in software development and in selecting the right process for a common process in teams. As said before an SPrL resembles an SPL in many ways as it is an approach of creating a process family and is a SPL in the process domain [65].

A SPrL is a set of processes that are already common for certain problem domains and if used in a company, the processes are mostly in place already [83]. In [99] the authors explain that large parts of the processes used can be reused with a SPrL, as opposed to a classical approach where the individual processes are defined for each project. The process that is used as a standard for a large company has to be generic as much as possible but also very detailed to support the daily work of the development department. The goal of a SPrL is to reduce the complexity and with it the effort to manage such processes in a company. The processes within an organization are often redundant and could be reused if one department or team has the knowledge that the process already exists or that a similar problem has been resolved with a certain process. This redundancy leads to unnecessary use of resources to provide a tool or methodology that has to be maintained. Here the SPrL comes into play, as it helps to reduce this mess. Another approach is to use an software process improvement approach, but as those are generic standards there are mostly sub-optimal if not tailored to the specific needs of the organization [69].

As described in [83] a SPrL needs a *Software Process Line Architecture (SPLA)* to get the variation and also the commonalities of different processes. The core processes and its variants are used to be a basis for every specialized process that is used for a certain project. The commonality in a PLA is shown with the core process, which consists of the common parts of the different processes. Variants and variation points build the variability, where these points are characteristics of a specific project. When a new specialized but similar project is done it is possible by combining, extending and reusing the core process and the variants to get a new specialized process. The set of variabilities and similarities of the processes that exist in an SPrL make them what they are [99] and is derived from one central reusable structure. This is what an SPLA is, and it includes variation points and optionalities. Optionality refers to the ability to express elements as optional or as mandatory. Additionally, dependencies can define if a type includes or excludes another type [99].

If we look at an SPrL again the processes that are generated with an SPrL need to comply with the defined characteristics. Those can be compliant with maturity models and life cycle models which are used. In this context the software process line engineering, which resemble SPLE from SPLs have to be mentioned, as those consist of a set of strategies and systematic approaches to build, apply and manage the processes of an SPrL [99].

In [83] an approach to build up a PLA from bottom up is presented. This approach can be used in this work as well as different processes can be gathered when performing the interviews. From these processes the common parts can be extracted and the commonalities can be analysed. Additionally, an SPrL as described in [61] could be used to describe the test process and define how an SPrL in this domain could look like. With the interview results it is checked wether an SPrL makes sense in the found processes or not. This is also done with the use case at Willhaben, where it's analysed how such an SPrL for testing should look like. The commonalities and variabilities that form an SPLA have to be described as well, which is done by checking the needs of the different people in a team, which in this case means, that we use the interviews and surveys for this. In [99] the focus lays on applying an SPrL to Scrum in combination with CMMI. This leads to the question if such an approach can be applied to SPICE and other agile processes too. This is answered in this work with RQ3, where the question is asked how an adapted software process could look like and how it changes when we use SPrLs for it.

CHAPTER 4

Illustrative Use Case

At Willhaben there are many development teams that are working on different products for the website and on features that help to produce a better user experience on the one hand and on the other hand more financial flow throughout the system. There are many critical parts that either influence the users directly or the paying customers that Willhaben has. To reduce the risk of shipping a production bug that has a critical outcome on one side or the other, Willhaben has a *Quality Assurance (QA)* team which tries to test all of the new features and products that are developed. Before 2014 Willhaben ended up testing the whole development cycle manually with the help of unit and integration tests, which meant a lot of regression tests and many repeated steps for the testers, but yet some sort of automated software testing to support the manual testers. These testers do provide a certain amount of security, because they locate and detect bugs before the final deployment on the production system. The QA-team was doing feature tests, retesting of bugs that were reported, did second level support as well and did a lot of regression tests before the release. This was one of the reasons no blocker was released on the production system. But because the complexity of the system in place increased, the regression tests took longer every time they were done. The developers had unit Tests and integration tests in place and also some smoke tests to test the availability of important components. The test coverage was okay for the most part, but some areas of business relevant code were not thoroughly tested.

Since 2014 Willhaben additionally added a *Test Automation (TA)* team which is part of the QA-team. This team, which consisted of two people, developed a first set of end-to-end tests in selenium for the desktop variant of the website. The thought behind this was that the amount of time spent on regression tests by the manual testers should be decreased while the coverage of the end-to-end tests should increase. This also should have had an effect on the deployment cycle from the company as it decreased the time between deployments on the desktop product, because the manual testers could spend more time on testing new features. This should have resulted in faster reported bugs,

which then could be fixed by the developers. This would also decrease the amount of blockers or critical bugs in production and therefore a decrease of lost money.

As the first set of tests was written a few things happened. The manual testers relied on the TA, but not as much as expected, because the tests were high in maintenance and had to be fixed more than once a month. As a second outcome the developers got affected in the development of the website and new features. They had to change the programming dogma, as they had to add identifiers for the TA. The developers didn't always know where they implemented unique identifiers or constructs on the website that could be accessed by the end-to-end tests. This lead to some refactorings in the software and the developers had more work than before, as they had to think about TA as well. Here the term TA is only for the development of end-to-end tests and not for unit or integration tests.

The focus of this work is to provide a way to automate the test process in such a way that the test process is always the same for every new project that is rolled out. The test process should include the developers, the QA as well as the TA team. Another focus are best practices that should be implemented when the test process is applied. These best practices can then be used in other companies that are working in the same field of work. Another focus is to provide a basis to check if a *Software Process Line (SPrL)* makes sense to be built for Willhaben and other companies as well and how this SPrL can support the test process that is implemented or adapted. Another focus is to identify an improvement strategy which could be implemented and used in a global team. This improvement strategy can use the best practices that are identified and standardize them to be applied to different teams. For this the Schibsted companies are used for a use case and Willhaben specifically. Therefore the different companies and the team structures are explained. Only those that are ad marketplaces and are included in Schibsted are listed here, this is done for simplicity and keeping the work in a manageable size.

Schibsted is a worldwide operating company which has a lot of sub-companies, which are based in various fields of work. There are the newspapers and media houses on the one side and the classifieds websites or marketplaces on the other. In this work only the marketplaces are compared in their team structure and their approach to testing in general and end-to-end testing specifically.

There are the six big players inside the Schibsted marketplaces, which consist of Finn, Blocket, Schibsted Spain, Leboncoin, Subito and Willhaben. These are the major ones, also called the big six, but there are marketplaces in 22 countries all over the globe. These marketplaces all have in common that they are only regionally active and all have their own implementation. Because of the narrow field they are operating in, they all have similarities in the structure of the site and the basic functionalities. The reason for this many different marketplaces is that Schibsted wanted to create a regional experience for the customers. This led to the overlapping features and structural commonalities.

As of now the cooperation is strengthened as more components are used for all the web sites and therefore the same features are rolled out in the different regions, but still with

specific limitations to every region.

The big six are the established marketplaces that are the number one marketplace in the corresponding country. This is Finn in Norway, Blocket in Sweden, Subito in Italy, Leboncoin in France, Willhaben in Austria and Schibsted Spain, which consists of more than one marketplace, but is organized under this name. These six are the most interesting ones for this work as the structure is well developed and the teams have their responsibilities in the company. Willhaben will be used as an example for all the companies compared because it would take too long to explain every single one, even if only the big ones are explained. The smaller companies are also taken under consideration, especially for the survey and the interviews, as those aren't as well structured as the established one.

Willhaben for example has eight teams right now, six development teams, one operation and one QA team. These teams are working for the *Product Management (PM)* team which prioritizes features and sets out the path for the *Product Development (PD)*. As said before in 4 the QA team consists of manual testers, which execute regression tests, feature tests, checking the logs and much more, and TA engineers. Operations are there for maintaining the main servers as well as the test servers and monitoring the deployments and roll-outs of features.

The other companies are larger in comparison to Willhaben, but are still comparable. More information on the structure and the different companies owned by Schibsted can be found on the official website of Schibsted [1].

One of the main companies that will be taken into account is Willhaben, which is one of the sub companies located in Austria. It is the biggest marketplace in Austria and consists of about 200 people in total. This number includes all employees which work for Willhaben. About sixty of these people are in product development. There is additionally a group of twelve, that are in quality assurance, which includes TA as well. Willhaben will be used for the use case, as an example of how the process could look like and what advantages come from test automation in the sample company.

[1] Marketplaces Schibsted; www.schibsted.com/en/Marketplaces/ Accessed: 2018-06-28

CHAPTER 5

Research Questions

Here the research questions that were previously found through the research gap in the state of the art will be explained. Additionally the influence from the problem statement is taken into account and the use case at Willhaben. At last the process how the research questions will be answered is presented and how they are influenced by a software process line. As mentioned in the introduction, we want to find best practices for software testing in general and for E2E testing in particular. This is mainly done through *research questions (RQ)* 1 and 2, where RQ 2 focuses strongly on the E2E tests and how specific test requirements have to look like. If you look at the second target, it becomes clear that it is exactly answered by RQ 3. In this question we will look at the best way to set up and use a test process. In addition, by using a software process line, an approach is provided as to what a standardized approach might look like. The challenges faced in this work are shown in figure 5.1.

5.1 Best Practices for Software Testing

Best practices can be useful in many different situations. For this work, we use RQ1 to look at which best practices should be used for test automation and what they should look like. They serve to better understand problems and to show, through use cases, why one should tend to them. In software testing there are many different frameworks, ways to test and programming languages to use as stated in section 2.2. This diversity makes it easy to lose track of the ways in which you can solve a problem. Best practices can help by using existing experience to find the best solution.

The research question - RQ.1: **What are best practices for software testing in context of processes, organizational structures and technology?** - arises when we look at the current state of practice at many companies that have more than one team, as they normally have very different approaches on software testing. This can be seen at Schibsted, where many sub-companies which work in the same field are using different

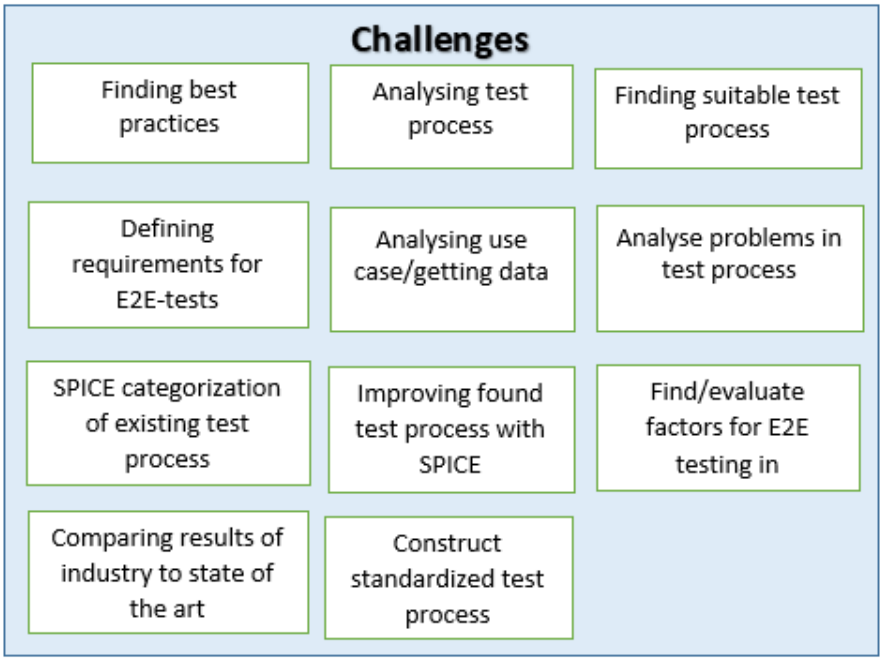

Figure 5.1: Challenges within this work

processes, organizational structures for the teams and different technology. In 2.2 and the following sections a certain picture is drawn for best practices in software testing, what to automate, how to choose a process and how to improve these processes. The researches such as [84, 58] show that much effort has to be put into software processes to be implemented and used correctly in an organisation. Software testing in general is described in [20, 22, 21] and it shows that there are many approaches how to do software testing correctly. Here the test automation pyramid from [48] shows the ideal distribution of the usage of the different testing methods, which are explained in [11, 13, 17] and in many other researches that were listed in section 2.2.2 and 2.2.

Different approaches are put together and a catalogue of improvements is written down to get an overview what could be improved in the future. The results for this catalogue come from the test expert interviews and the case study. The results will be compared to the current state of practice and point out the biggest areas of improvement.

5.2 Specific Requirements E2E-Tests

Finding the exact requirements for *End-to-End (E2E)* testing in different companies is a task that becomes extensive and complex due to the sheer number of aspects that need to be considered. But this task has to be done to create efficient and good E2E tests and to establish them in a company in the long run. For this you need the right selection of

tools, the tests must be understandable, maintainable and meaningful and the results of a test must also be usable.

With the question - RQ.2: **What are specific requirements for end to end (E2E) Tests?** - the E2E tests are analysed and to find out what is required that these tests are implemented efficiently and effectively. Just because E2E tests are developed doesn't mean that they are useful for the project and further for the company as was explained in 2.2.2. The different testing frameworks that are mentioned in section 2.3 are another part of comparing requirements for E2E as not every framework is suitable for every test case and project. There are different kinds of E2E tests as stated in [17, 8] and they can be implemented differently as stated in [24]. For the requirements of E2E tests it's important to know which purpose these tests fulfill and what the main problem is which should be solved with these kinds of tests [24, 17, 8].

This question is answered with the case study, the survey and the test expert interviews, as the requirements for successful usage of E2E tests are defined and the results are compared with the current state of art as stated in section 2.2.2 and 2.2.

5.3 Improve an Software Test Process Approach

Changing a test process brings opportunities on the one hand and risks on the other, of course, as you deviate from your usual path. The changes, if well considered, can lead to a strong positive change, which on the one hand makes resources easier to plan and also opens up new paths and opportunities for the employees. The re-evaluation of an already existing process also provides insights into the problems that currently exist and whether these could not be avoided by adapting the process.

Willhaben currently has a test process, but the question - RQ.3: **How can we establish an Adapted Software Test Process Approach at Willhaben?** - tackles if and how a more sophisticated approach can be taken. This approach can then be used as a blueprint for every new project at Willhaben and here the software process line can help with this. There are different test processes mentioned in researches, especially regarding agile methods. *Test Driven Development (TDD)* is one possibility [57] for a development practice to improve the software test process. Another approach is *Behavior Driven Development (BDD)* which is described in [30], which also can improve the test process. How a test process improvement could look like is described in [33]. Different test processes are described in [34]. The test process is a focus of this work and the described test processes in section 2.1 are used to check if the test process at Willhaben resembles the current state of the art.

The software improvement approaches that were mentioned in section 3 are good starting points and can be compared to the status quo. In this work the *Software Process Improvement and Capability (SPICE)* approach is used, which have specifical areas of improvement for testing [82]. Other improvement approaches are ISO9001 or *Capability Maturity Model Integration (CMMI)* which are more general than SPICE as described

in [73, 77]. Additionally a *Software Process Line (SPrL)* could be used for analysing the current test process and if it is possible to use a nSPrL to improve the overall performance of the test process. In section 3.1 an SPrL in theory is described. SPrL in global companies which are described in [70, 83] are used in this work as Schibsted has many companies around the globe. A general process that could be used for every software development team at Willhaben, as described in [83] could be considered as well. In [61] an example for an SPrL is described and how their approach on SPrL changes some of the previous approaches on this topic.

The test expert interviews, the survey results from Willhaben employees and the case study answers the question partly as well. The found results can be taken into the catalogue of improvements and be a use case for anyone that wants to establish a test process in another company, that is trying to use test automation.

CHAPTER 6

Solution Approach

In this chapter the various methodologies which are used in this work are explained and the way it is applied to the different problems that were previously mentioned. In addition, the way to the current status quo is explained and how the situation changes when using an *Software Process Line (SPrL)*. Figure 5.1 shows the challenges arising from the research questions. Those are addressed by various methods as explained in this section.

The current process in Willhaben is a grown process that has been improved or changed by different approaches. The current approach is based on Scrum, but it has been adapted to Willhaben's needs, mainly in terms of organisational issues, as there is no dedicated Scum Master to take care of each team. In addition, a software tester is not a permanent member of the team, except in some specific teams. This means that a tester always changes teams when he is needed more urgently elsewhere.

The design science methodology to identify the needs and requirements of the RQ's is used and as described in [91] needs to have a design problem as well as at least one knowledge question that gets information about the world. The research questions are the basis and the description of the problem that we want to improve. But those research questions are knowledge questions as well, as they ask for knowledge about the world as described in [91]. In figure 6.1 a slightly modified framework for design science is shown, as there are no stake holders for this work, and because of that no budget and designs that have to be provided. In [91] it is specifically stated that the stakeholder is someone that pays for the research.

The knowledge that is gathered through the process is separated into two categories by the author of [91]. Those are prior knowledge and posterior knowledge. Everything related to the literature that was found before in chapter 2 is prior knowledge, but the used methods in chapter 8 and chapter 9 are posterior knowledge that was gathered through the research done.

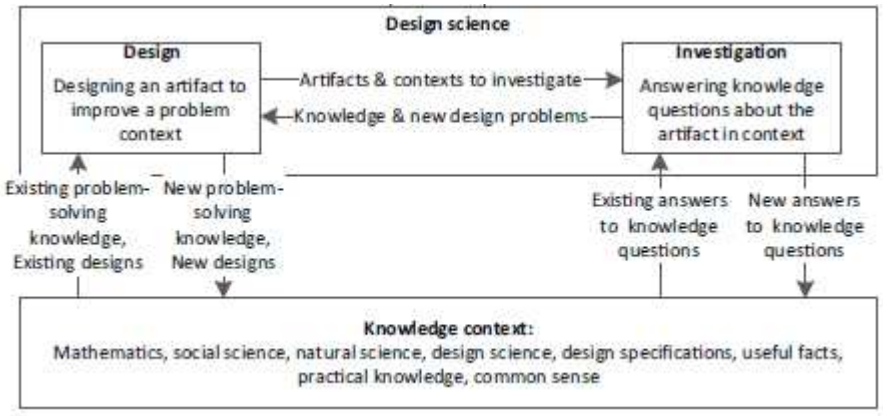

Figure 6.1: Adapted design science framework from [91]

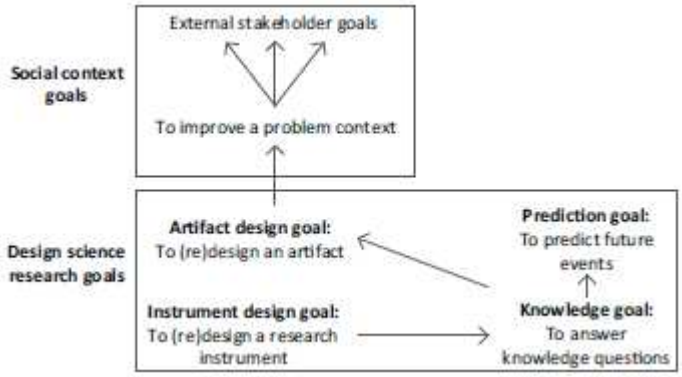

Figure 6.2: "Goal structure of a design science research project" R. J. Wieringa, [91]

There are goals to reach through the design science approach, which are described in [91]. Those goals either concern the beliefs of the past, present and future phenomena or are design or generally speaking improvement goals. As described before it was necessary to answer some knowledge questions, which was done with the interviews as well as the survey. Those two methods are described as instrument design goals, which are classified as the lowest-level design goals. This can be seen in figure 6.2.

The knowledge goal is solved by the methods, as they provide knowledge that is gathered. From the results of the knowledge goal and in this work with the help of *Software Process Improvement and Capability (SPICE)* the artifact design goal can be approached. The goal of this work is to provide a catalogue of improvements or best practices that can be used for software testing with the focus on automated software testing. As the authors state in [91] research questions are nothing else than a design problem with a different template.

As the main methodology of improving the software testing process SPICE is used in

this work. This process improvement approach is described in subsection 3.0.4. The main purpose is to measure on which capability level a certain software process is and how it can be improved so that it reaches the next level. This approach is used to get information about the world to answer a knowledge question that was asked before with the design science methodology.

To find out what the state of the art in the different organizations is, a survey that was sent out to every single company, was used. This helps to get information about the best practices for software testing, which is required for RQ1. This survey follows the approach of [87], by providing the same chance of entry to everyone in the companies, that were selected for this. The approach here was to take a survey which questions the ongoing process for the current state of practice. This was done by reaching out to work colleagues which are situated around the world and in different teams as well. As stated by [87] a survey selects participants of a group so that everyone of the selected group has the same chance to be selected and participate in the survey. This was done by sending the email to every *Chief Technology Officer (CTO)* of every company with the request to redirect it to all employees that are working in product & tech. These doesn't only include developers and testers, but also product managers and others as well. Those people don't have an answer to every question, but instead offer a different view on some questions. The questions are categorized in six different blocks, all regarding distinct aspects of testing, personal information about the person which is doing the survey or the survey itself. The questions offer either an open text answer or for example a selection of the most popular tools that are currently used as state of the art, chapter 2, with an extra field for other answers that weren't covered with the presented ones. The wording of the question was chosen to be as precise as possible and help to clarify what was expected. This means that the survey was designed to agree with [87].

The SPrL was an approach to find a general test process which can be applied to the whole organisation of Willhaben and even further in a global matter. As [99] shows an SPrL which is applied to Scrum in combination with *Capability Maturity Model Integration (CMMI)* is an option, but as said before we use SPICE as our model to improve the software process. Therefore we can use the shown methods from [99] and apply them, where it's possible or modify them for this work. The identified areas of improvement are then analysed and an SPrL is applied to see to what extent a standardised test process can be used in the company. In addition, the suggestions for improvement are collected and presented.

CHAPTER 7

Initial Analysis at Willhaben

This chapter describes the specific use case at Willhaben and how the current state of practice is. The current state is then analysed by using some specific parts of *Software Process Improvement and Capability (SPICE)*. After that it is checked if a test process improvement can be achieved and if a *Software Process Line (SPrL)* can support the improvement of the test process. This SPrL is then looked into if it could be adapted to be used globally. This is done by using the test expert interviews of various other companies and by comparing those test processes to the current one that is used at Willhaben.

7.1 Current State of Practice

At Willhaben there is already a certain test process in place, which grew with the development of the company. It got adapted every time the company grew significantly and new changes in the web page were introduced. This leads to the current test and development process, which as stated by the interview partners from Willhaben, starts with a specific *Objective Key Result (OKR)* that has the focus of most *Product Development (PD)*. The project behind the OKR is prioritized and a team consisting of developers, a project owner and sometimes a dedicated tester works on the implementation of the features.

When the first features are done, Unit tests and integration tests are written for them, therefore there is no *Test-Driven Development (TDD)*, which was mentioned earlier in chapter 2 and isn't as the state of the practice in [57] suggests. A short development cycle is the basis for TDD, which is currently not measured in a metric at Willhaben. Such a metric is planned to be able to see how much time is needed from a feature ticket to releasing it. Additionally, this would show where a bottleneck is located and if the current process has to be adapted. The development cycle is one part that could be improved, but as some of the interview partners said, the practicability of TDD isn't

always given in an ever changing environment and sometimes only suitable in a research context. This part of the test process is mainly done by developers, as they have to write those tests. Some API tests are written by the *Quality Assurance (QA)* team, but those aren't the main focus of this team.

The test process then continuous but is now in the remit of the tester and QA, which supports the dedicated tester when bigger parts of the software are released or refactorings of code that influences many components are done. If no dedicated tester is assigned to a team, the QA team has testers that are able to test in every team and help each other if the demand is higher. The testers do feature, usability, black box and other tests as stated from [7, 11, 8], but no regression tests as those are done by the automated *End-to-End (E2E)* tests. As shown later in the test expert interview results these tests help to enable a faster release cycle as tedious and repetitive work is removed from the workload of the manual testers.

The automated regression tests are run every day either at night or early in the morning, which ensures that developers or testers and additionally build processes on the *Continuous Deployment (CD)* system aren't disturbed. The tests can fail if a precondition isn't met, such as database restores or a successful deployment of the test system. This can happen when faulty code is committed or a procedure fails, but isn't a problem, as the tests can be started manually again, after the failure is resolved and fixed. When this happens the tests are executed during the day, which can cause problems as well. If a deployment is started during the test run, the test environment isn't available for the tests and therefore the tests fail. Another side effect is that builds that are started on the CD system are executed slower and have a higher duration which can lead to failures in the build process.

The E2E tests do run for most of the projects, but as new projects and platforms are introduced the E2E tests have to be implemented for those systems as well. Projects which are currently not automated are Android and iOS, which are automated with Appium, which is described in [23] and in section 2.3.4. The automation of the apps is focused, as it is a blindspot in the test coverage and this could lead to a bad user experience, as the manual testers can't do full regression tests for every release cycle. Other projects aren't that important to be automated, as they have a smaller impact at the broad user base and therefore can be put aside, but are being planned for the future.

As stated by [18] test automation is time consuming, especially when a foundation of a variety of tests is developed and then development goes further and parts of the already tested code are changed. When this happens the E2E development team has to fix the existing tests as fast as possible, as those failing tests don't provide any information about the current state of the *System Under Test (SUT)*. As the E2E tests are used instead of manual regression tests, it's important to get reliable test results and not false positives.

Before the release the tests are started manually again and the test results are checked intensely to provide a final approvement. If there are many bugs found in this stage,

the release is stopped and the bugs have to be fixed. Then the E2E tests are started again. This cycle can occur more than once to provide the best possible version for the release. Here false positives can be a problem as they can mask bugs that lie behind the failure. Because of that the manual testers check manually if a bug is hidden. If they find nothing, the release is approved and is deployed to production.

There are special cases were smoke tests or stress tests are executed, especially if the hardware of the servers is changed, but this rarely happens and isn't part of the standard deployment and therefore test process.

This test process is nearly the same for every single project that is carried out, the only difference is the agile process that is used by the team. Some use some sort of Scrum, while others use an adapted form of Kanban. Because of that only one test process from one team will be analyzed and improved, as the others are similar to it and can be adapted slightly to fit the other teams.

The teams use JIRA as a bug report tool and Confluence for the specification of the features that are in the respective release. Bugs are rated in five different impact risk groups, which are blocker, critical, major, minor and trivial and they are fixed accordingly. Trivial bugs can potentially be released as they affect either a small number of people with a minor bug, such as a wrong encoding or are minor graphical disturbances. Everything from major onwards will be a stop for a potential release as the danger of a huge impact is too big.

7.2 SPICE Assessment of Selected Process Areas

In this section the process improvement approach for the test process is described and how it could be applied to the current process at Willhaben. For this the test process is analyzed and checked on which level the test process currently is.

SPICE is a process improvement approach as described in 3.0.4 and in [82, 80]. It is developed by ISO and has 6 capability levels that can be achieved. The software test process of Willhaben will be rated with the criteria of SPICE and then a process improvement will be discussed. As SPICE doesn't have one test process it will be split into the different parts that touch the test processes, such as the quality assurance process. The main focus of this work is shown in 7.1.

There are different process groups from SPICE which are interesting for this work. First and foremost the *Software Engineering (SWE)* process group, which has testing processes included and is the process group for software engineering. This group is part of the primary life cycle category as described in [82] and in section 3.0.4. The second process group is the *Supporting (SUP)* process group which is part of the supporting life cycle category, and has quality assurance as a process group listed. The process improvement from the organizational life cycle category in the *Process Improvement (PIM)* process group is an additional focus, as the goal is to improve the test process.

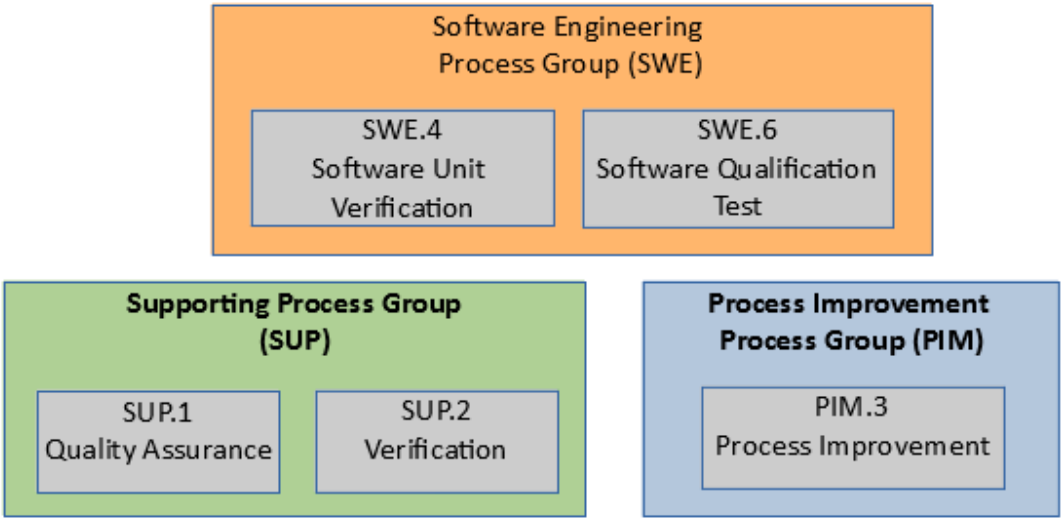

Figure 7.1: Selected process areas from SPICE

As said before the capability levels with the process attributes will be used to rate the test process and the rating scale will help with it as well. There are 3 methods to rate a process and the method will be chosen accordingly, as the rating method can be changed for every class, scope and context [82]. Method 1 uses both the process attribute and the process attribute outcome level, as well as the type of aggregation rating of each process, instead of method 2 and 3 which only use the process attribute level.

In SPICE [82] the different process groups are described in detail, what the process does, which process outcomes can be expected, which base practices are needed and which output work products are necessary. With this it is easier to rate the processes which impact the test process. To achieve at least level 1 of the process capability the process has to "achieve the intent of the base practices" S. Automotive, [82]. These *Base Practices (BP)* are shown in the following sections as well as the achieved capability level.

7.2.1 SWE: Software Unit Verification

The first process that is analysed is the software unit verification process. It is a process to verify software units and check if they are compliant to software detailed design and the non-functional software requirements. The process outcomes are described in [82] for each of the further mentioned processes.

The current status of the software unit verification process at Willhaben is that a strategy to verify the software units is in place, with unit tests that are written for the features that are developed. Regression tests for the regression strategy are clearly defined and a regression strategy is implemented. The criteria for the software units are specified accordingly and the unit tests meet the requirements of the feature they are testing. The

requirements are written down by the product management in JIRA with additional information. This is the second BP in this process.

The previously performed criteria for verification have to be used in the static verification of the software units, which is done by having *Pull Requests (PR)* that have to be checked from another developer. In these PR's the code is also checked if it aligns with the coding standards, that have to be applied to every feature.

In BP 4 unit tests are the center of attention, with recording the results and logs. Because there are unit tests developed and executed whenever a build is done, this BP is done at Willhaben. The tests are recorded in the *Continuous Integration (CI)* tool where all the builds are logged and the results are published.

The bidirectional traceability is achieved by analysis of the tests with different tools that are used. Additionally code coverage of the tests is measured and usefulness of a test is verified. The PR review is also a tool for tracing the usage of unit tests for all the features. This ensures consistency as well, as the developers have a tool to track failures back.

The last BP is summarizing and communicating the results of the unit test and the static verification results. This is done via email for all the builds that are executed as well as having build and version tool to look into specific commits and features.

As all BP are achieved and therefore the generic resources are also achieved the process can be rated. This means that the software unit verification process has reached capability level 1, as this is the only requirement for the *General Practice (GP)* of the capability level 1. This is achieved by all of the following processes as well.

For the second capability level there have to be more GP's to be achieved, as well as *Process Attributes (PA)*'s. When checking those GP's against the process at Willhaben the following occurs: objectives for the performance are identified and achieved as specification tickets and therefore artefacts for the unit tests are written in the specified time constraints, and the tests are measured and checked if they are conform with the test process. The process cycle time as well as the usage of resources is defined when a sprint is done and the available time is clearly specified for the unit tests. For the second GP a plan for the fulfilment of the identified objectives has to be defined. This is done when the sprint plan is defined, or the development cycle when a team uses Kanban as a development process. In the BP key milestones should be defined, which is done as well in the different meetings of the teams. Additionally a process work product review has to be done, which is done via jour fixes at Willhaben and reviews of the sprints, after a deployment was finished. There the process performance is discussed as well. The process performance is monitored with the completion of the feature tickets, which include the step to develop a unit test as well. It is also monitored with test coverage metrics and peer reviews, as well as pull request reviews. If the process has to be adjusted it is done, as the project owners and product managers are in direct contact to the developers as well as the team lead. This enables a fast reaction if the process doesn't fulfil all the requirements of the project and if someone identifies the need for a change. The agile

coach of Willhaben is in every team to define new processes and check if the process is performed correctly. For the 6th GP the available resources for performing the process accordingly have to be identified, prepared and made. This is done at the management level, such as a switch from human resources from one team to another, especially if a project that is carried out by a specific team gets a higher priority. For these switches a training or mentoring is needed, which is done in the teams themselves, to prepare the new team member for the used programming language or etc. For the last GP interfaces for the communication, responsibilities and process performance measuring are needed. Those are provided by the company for every team.

The second PA that has to be largely achieved is the work product management. Here the work products are the focus and the following GPs have to be done. Work products have to be defined, as well as requirements for documentation and control have to be in place. This is done largely in Jira, as well as in Confluence, where feature and therefore test specifications are written down. At Willhaben a feature has to look a certain way, which are the requirements for the documentation. When a unit test is written it is reviewed afterwards and checked if it deviates from the coding standards.

This means that the software unit verification process has achieved capability level 2. The third capability level is the established process. Here a standardized process has to be defined and maintained for the first GP. This is done by using Kanban and a sort of Scrum like process for the different teams at Willhaben. The different roles of Kanban and Scrum, as described in [36, 35] are fulfilled by people at Willhaben. The infrastructure and the work environment in the company are given as tools for monitoring the test coverage and development of unit tests are provided. The generic resources that have to be produced by the process are produced, as training material for example is provided and a process infrastructure is defined.

As the second PA for level 3 the process deployment has to be achieved. First a defined process that is derived from the standard process has to be deployed, which is as said before done with Scrum and Kanban. The roles in this process are determined and every role that has to be set is set accordingly. The infrastructure as well as resources are given, and the competencies are ensured. As the last GP data has to be collected about the used process, which is done via different monitoring, lead time of the tickets or increase of coverage of the tests, when a feature is completed. As these PA's and its corresponding GP's are achieved, it defines that the capability level 3 has been achieved.

The predictable process is capability level 4. For this the business goals have to be identified for the quantitatively measured process. This isn't fully achieved at Willhaben, as business goals aren't always related to the test process and tests aren't necessarily seen as a necessity for the achievement of business goals. The risks of doing so are identified but not every risk is communicated to the product management.

7.2.2 SWE: Software Qualification Test

The next process that is analysed is the software qualification test process, which is about regression testing and having a release plan for the software that is developed.

There is a software qualification test strategy in place at Willhaben with regression tests that are executed manually and automated. Those tests are consistent throughout all software projects that are developed at Willhaben, as well as aligned with the project plan. Before a release there is a specific procedure that has to be executed, before the release is accepted. This is the first base practice (BP), which produces a test plan.

The second BP that has to be done is to develop specifications for software qualification tests. This is done when a new feature is written, because after that the specifications for test cases and test hints are produced. This BP generates the test specification with BP 3. BP 3 stands for test cases that are selected to be executed and they must be aligned to the software test strategy and the release plan. This is done with the manual tests, as well as with the automated E2E tests because of time constraints on the one hand and the release cycle on the other hand.

The integrated software is tested with the selected test classes by the manual testers, which choose which test is necessary as said before. The test results are recorded when integration tests are done and when E2E tests are executed. The manual testers use JIRA as a recording tool.

BP 5 is about establishing bidirectional traceability which is as said before achieved with different tools. This is the same for BP 6, where JIRA is used to trace bugs and features.

The test results are again summarized in the build tool, as well as with email and communicated via email. Every one that has interest in the different projects has the ability to sign up for the results. Slack is another option to get the information, as bots are used to get the information to the right people.

As before in subsection 7.2.1 the capability level is checked. This process is at least at capability level 2, as all BP are achieved. To get to level 2 the performance management and the work product management process attribute have to be completed. For the first PA the objectives for the performance of this process have to be identified, which is done by the QA department. The regression tests are monitored as well as documented and a release plan is done, and for those two elements performance measurements are defined such as different metrics on how fast the regression tests are done and how often they are executed. This helps to plan for the fulfilment of the performance of the identified objectives. As said before the tests and the releases are monitored and checked if they meet the requirements. If an issue is found, such as slower regression tests because of a slower environment, this is fixed, either from the side of the environment or the hardware for the regression tests is optimized. The software qualification test process has determined responsibilities and personnel which has to perform the process. Resources that are needed to run the process at its optimum are allocated and the included parties are informed and communication is done between them. For the second

PA the requirements for the work products have to be defined. When the test process is checked at Willhaben these requirements are defined by the team lead, as well as by the QA team itself. Reviews are done internally through the team, which is written down as a standard in Confluence and in the Willhaben wiki. The documentation requirements are also defined, this includes for example a javadoc for the automated E2E tests. Refinements for the work products are done in various ways, either in meetings or in workshops where new information and inputs are gathered. So all of the GP of the two separate PA's are done, therefore this process is at capability level 2.

For level 3 the process has to be an established process, which means to have a standard process, which is in place, as there is a certain process to do regression testing. This includes a development process, where Jira tickets are prioritized as well as test results are checked. The second PA for capability level 3 is about the process deployment. But let's look at the first PA, that has to be achieved. The standard process for a regression test is defined in the QA and interacts with other teams, which are introduced to the process. The hardware to write tests and the environments to run those tests are declared and available for the persons working with the tests. The standard process is monitored and re-evaluated when something goes wrong. The generic resources for this level are:

- Process modeling methods/tools
- Training material and courses
- Resource management system
- Audit and trend analysis tool
- Process monitoring method

Those are the outcomes if the process is fully achieved for the first PA, which is the case at Willhaben. For the second PA the process deployment has to be achieved. This includes tailoring the standard process to the needs of the team, which is done with two processes in the QA team. The automated regression tests use the page object pattern as a development process and a pull request review system to check if the feature is implemented correctly and checks against all necessary paths. The personnel is a dedicated group and responsibilities for the whole test process are declared. This is done through all of the different processes that are checked in this work. The team is instructed, has the right trainings and/or educational levels and has experience in the field of work. The infrastructure is given with the office as well as resources that are provided from the company. As all of the test results are saved the process can be re-evaluated and checked if it is still suitable for the test process.

The predictable process is capability level 4, which, when fully achieved for the PA's, result in:

- "The process is aligned with quantitative business goals"

- "Process information needs in support of relevant defined quantitative business goals are established"
- "Process measurement objectives are derived from process information needs"
- "Measurable relationships between process elements that contribute to the process performance are identified"
- "Quantitative objectives for process performance in support of relevant business goals are established"
- "Appropriate measures and frequency of measurement are identified and defined in line with process measurement objectives and quantitative objectives for process performance"
- "Results of measurement are collected, validated and reported in order to monitor the extent to which the quantitative objectives for process performance are met"
- "Techniques for analyzing the collected data are selected"
- "Assignable causes of process variation are determined through analysis of the collected data"
- "Distributions that characterize the performance of the process are established"
- "Corrective actions are taken to address assignable causes of variation"
- "Separate distributions are established (as necessary) for analyzing the process under the influence of assignable causes of variation" from S. Automotive, [82]

The test process has to be aligned with the business goals, which is to provide the best quality product and provide the best platform to sell classified advertisements. The stakeholders for the test process are identified and needed information is provided to those stakeholders. The process is measured with the coverage and with the time spent for regression testing. Additionally the release plan is checked for efficiency and if it is possible to keep up the schedule. For the measurements if the quantitative objective has been met, monitoring is deployed to get all the necessary data. The OKR's for the test process and PD are used to check if there are deviations from the defined objectives. If there are deviations then counter measures are taken to prevent them or to change the process in a way it's impossible to introduce them again. If something is an explainable variation which causes a change in the process performance it can be identified with the monitoring and the progress of the OKR's.

7.2.3 SUP: Quality Assurance

The first process of the SUP is quality assurance. This process checks if the work products and processes meet the requirements of the predefined provisions and plans and if a miss match is found, a solution is provided for that. The BP that have to be covered, are explained and checked if they are done at Willhaben.

BP 1 is to develop a project quality assurance strategy [82]. A strategy for quality assurance is done by the team lead of QA which coordinates the tasks that have to be done for ensuring the products' quality standards. There are certain standards and criteria that have to be met, before something new is released and even in the development phase there are certain factors. Reviews on features and meetings about getting better quality standards are done, as well as reports that are written for management. At Willhaben the goals for the product development are written down in objectives and one of those objectives is a QA one.

The assurance of the quality of the work products is always controlled, as said before. The testers are well instructed on what has to be done. The automated tests which are developed by developers and testers are supporting the efforts that are put in from the manual testers and provide a better understanding which areas fail in a feature development.

BP 3 is about assuring the quality of the process activities. The processes in QA are controlled by having standups in the teams, where problems with a certain process are talked about. When a new process is introduced a meeting about it is done and the results are documented. In test automation a documentation about the existing tests is kept and reviewed to ensure that critical paths are still covered. This also helps with reporting to management what was done and how the results turned out. The relevant people get the information about QA, what has to be changed to keep up with the development and which threats exist that could disrupt the development cycle because QA is trailing behind.

Re-evaluation of certain processes is done periodically to find out deviations or non conformance in a process. They are analyzed by the whole team or if something only affects and hinders only one person this is done by a face-to-face conversation.

BP 6 is about implementing an escalation mechanism which is in place at Willhaben, such as the JIRA tool, where for example a bug can be highly prioritized. Another way is to report the problems to the team leads and chief technical officer of the company. When this is done the problem is prioritized and solved. A hindering problem can also be addressed in the QA stand up and to get the attention on it from every QA member including the team lead.

The capability level 2 is reached, because the performance criteria are defined, planned and monitored, as well as the performance of the process itself. The personnel as described in [82] that performs the process are assigned, as it is the QA team and the developers as well that have to do quality assurance. The reporting between the different parties

is enabled, as the testers and developers talk to the responsible product manager if a feature isn't well enough specified, or they talk to each other on how a feature has to be tested as well as how the correct implementation should look like. The work products for the test process are defined and the documentation is done either via Jira, Confluence or the Wiki of Willhaben. After a work product is finished it is reviewed.

As said before the established process is the next step, which is having a standardized process with tailoring guidelines and deploying a defined process, which is either a standard process or a tailored standard process. The standard process for QA is defined as a mix of different test strategies, which include black box, white box, unit, integration and E2E testing. It is clearly defined what has to be done when a feature or a project has to be checked if they meet the required standards. For each project the standard process can deviate, but the core stays the same. The interactions with other processes such as the release process or the development process are defined and the roles and competencies for performing the process are defined as well. Suitable methods for monitoring the process are Jira-tickets which are closed, and test coverage, as well as the fulfilment of OKR's. The resources, in form of infrastructure and human resources are provided, assigned and the responsibilities are defined. This includes for example the determination of one tester for a single project, with a back up for safety. The results of testing are documented and this helps to make the QA process better.

7.2.4 SUP: Verification

The second process in the SUP group is verification. This process confirms if each work product of a process or a project meets the specified requirements.

The first BP is about the development of a verification strategy for activities, work products or processes. In [82] the authors suggest that peer reviews, audits or other verification methods and techniques should be used. This is done at Willhaben with developer jour fixes, pull request reviews, coding dojos and other activities to verify code and testing. With all of these strategies criteria for verification are needed and those are for example coding standards that help to verify pull requests.

The coding standards are written down and everyone that is writing code has to follow it. When something doesn't meet the requirements the pull request isn't accepted and improvement suggestions are given in the versioning tool.

Tests are also used for verification, so that problems can't slip through to the production system. If a bug is found in production it is tracked and reported. Additionally a review of it is done and it's decided if an automated test should be written for it, so that it doesn't appear again via regression.

The capability level for verification is at three, which, as the other processes analysed before, have to have a standard process and this process has to be deployed either as is or tailored to the specific needs, which is the case at Willhaben. The test process includes verification of the work products and the project itself, which is carried out in

peer reviews as well as feature completion and in the case of the test process if the bugs are closed and reviewed. The infrastructure is provided as said before from Willhaben with all the resources that are necessary. The people performing the verification process are assigned and have the authority as well as the responsibilities for the project and the work products. The data from the test results is gathered and after a project is deployed to production the information is analysed and checked if there are any parts that need improvements.

To reach the third stage of the capability levels the process has to be managed. This is fully achieved as the performance management and the product management PA and their respective GP are fully achieved.

7.2.5 PIM: Process Improvement

As the last process that is analysed we check the process improvement process of the PIM group, which checks the efficiency and effectiveness of the processes that are used in a company. In this work the focus is on the test process [82].

In the establish commitment base practice of this process the authors note that the process improvement process is a generic process and that it can be used for every process that is developed in a company [82]. BP 1 is achieved at Willhaben, as team members are introduced to new processes, meetings and therefore resources are provided from the company to get the most out of the process. Individual goals are set to get a higher commitment for the processes, or people are put into task forces that have the goal to improve a certain process.

As the second BP [82] defines identifying issues. This is done at Willhaben with goals in the OKR's and with the different task forces to tackle a specific problem. Additionally bug reports and problems that occurred are handled and analyzed so that a process can be improved. Production bugs and problems are additional sources for information and customer input in face to face interviews are used as well.

The establishment of process improvement goals is done in meetings and in overall goals to improve performance and effectiveness in certain areas. This does include other departments at Willhaben as well, such as digital advertisement. There are smaller teams that specifically evaluate the status of a current process as as well to support the creation of processes and setting a goal for those processes. The process improvements that are taken from all of these meetings and evaluations are then matched with the current goals and a prioritization is done.

When a process has to be changed all relevant parties are involved in the planning of the adapted process. This is done through various teams, but for one example the release process had to be adapted from the previous to the current process. Therefore operations as well as developers and testers made several meetings to improve the process. This is BP 5 and leads to BP 6 as the process was then implemented in the whole development department. The training is currently going on for this process, but other processes such as the test automation process was adapted as well over time.

For BP 7 the process improvement has to be confirmed and for this there are different tools available at Willhaben. Those are for example Jira, where the feature and bug reporting process was changed, or with Kibana, the monitoring system that is used. Additionally there are the OKR's where the achievement level of the key results is measured. This helps to keep track with a certain process that is improved. This also helps with communicating the improvements to the organization. As Willhaben has no customer per se for the developed products the internal departments and the global organization are more important to spread the improvements and end results.

As BP 9 from [82] states the results of the improved processes have to be evaluated and checked if they could be used somewhere else. This is done at Willhaben and even at a global level, as processes and experiences are shared between the different companies.

This process is, as defined by the capability levels in SPICE, at level 1 of the scale, as it clearly achieves all the BP as a GP for level 1. The second level isn't reached as no monitoring and performance measurements are implemented.

7.3 Test Process Improvements

In this section the possible test process improvements for the processes analysed before in section 7.2 are listed. The different processes are on certain levels of the process capabilities. Therefore each process will be looked at separately and how it could achieve the next capability level. Those process capability levels are shown in a table to get a better overview about them.

For the first analysed process the SWE.4 as it is called in [82], the capability level is 3. This means that there are still 2 levels to be achieved by this process. For this the GP of capability level 4 and 5 have to be achieved and fully completed. As said before the business goals are identified, but the relationship between those goals and the process aren't clarified, so that it isn't fully achieved, which is needed. Additionally the quantitative objectives aren't defined for the unit tests and the verification level of them. This could be improved by defining such goals and to implement a metric for quantitative objectives. From the generic resources the product and process measurement tools and results database aren't done at Willhaben. There are coverage metrics, that are used, but only occasionally. To achieve the fifth capability level innovation opportunities are the most improvable aspect, as there isn't much research on how to innovate unit testing in the company and how to implement such an innovation as well, as time constraints hinder these implementations. There are some teams that try to integrate new tools and programmatic approaches, but there is little to no time as feature requests are prioritized higher.

In the software qualification test process or SWE.6 the capability level is at level 4, with only the innovating process as the last level which isn't achieved. The analysis of the process to identify opportunities for innovation isn't done as it should be. Here the process could be improved, to check if a new test process for regression testing and a

new release plan would be better fitting for Willhaben. New technologies are checked but they aren't used as personnel restraints are limiting the process. Risks are identified but as stated by [82] they have to be evaluated if they could provide improvements.

The third process is quality assurance or SUP.1, which describes the testing level even further. Here the capability level 3 is reached. The fourth level isn't achieved as the second PA isn't achieved largely, as tools for analysing the process data aren't selected. Additionally there are no corrective actions taken to address assignable causes. There are only little statistical analysis tools or applications used for checking if the QA process drifts apart from the expectations. This can be improved and periodic checks should be implemented to get information faster if the test process has to be changed.

Verification or SUP.2 is the next process that can be improved in different areas according to it's capability level, which is three. For the next step the verification process has to be predictable. This isn't achieved as not every team has their verification process aligned with the business goals. Another GP of this level which isn't achieved at Willhaben is that process measures aren't identified for the achievement of the quantitative objectives. Different analysis techniques aren't used and defined, which has to be done for improving the capability level. Assignable causes aren't identified and therefore no variation can be found for the process. This has to be tackled at Willhaben to get a predictable process and then further improve it to an innovative process. The different development teams differ greatly in their respective processes as for example the mobile application team has a much further developed process for verification than other teams.

The last process which was analysed was the process improvement or PIM.3. Here we are on a lower capability level compared to the other processes, as it is only on level 1. There is no performance monitoring and the objectives for the performance of the process aren't identified. This means that not every GP is achieved for level 2. Monitoring the performance should be implemented as it is important to get the improvements of a process as fast as possible to check if it can be further improved. When the necessary steps for capability level 3 are checked, there is no standardized process to improve other processes and every newly introduced process at Willhaben has to be done nearly from scratch. The processes are documented but no basic process is kept in place for further improvements. Additionally there are no competences and roles defined for performing a new process that was adapted from the basic process. The process improvement process isn't standardized, but this could be done to improve the performance of other processes if a standard improvement process approach is found. Before this isn't done, the capability level 3 can't be reached.

In table 7.1 the various levels of the processes that were analysed through the research are shown. It's just a summary to get an overview quickly.

Table 7.2 shows the different areas that have been examined and additionally the possible vulnerabilities that should be fixed to get a better test process.

The weaknesses of the individual processes depend on which capability level was reached. The lower the level, the higher the potential for improvement and the risk that something

Process ID	Process name	Capability Level
SWE. 4	Software Unit Verification	3
SWE. 6	Software Qualification Test	4
SUP. 1	Quality Assurance	3
SUP. 2	Verification	3
PIM. 3	Process Improvement	1

Table 7.1: SPICE capability levels of the researched processes

Vulnerabilities:		
SWE.4	Quantitative objectives	Implement coverage metrics
	Research improvement methods	Prioritization of tests over features
SWE.6	Identify opportunities for innovation	Improving regression tests
	Way to improve recording	use new technology
SUP.1	select tools to analyse process data	
	stronger commitment to quality assurance in business goals	
SUP.2	align verification process	use collected data for assignable causes
	verification is neglected in business goals	
PIM.3	Implement process performance monitoring	
	standardized process for process improvement	

Figure 7.2: Identified vulnerabilities of each process

goes wrong in the process. The more regulated a process is, the better the risk can be assessed.

7.4 Software Process Line

Here the *Software Process Line (SPrL)* for the test process is researched, if it makes sense to use a SPrL at Willhaben and how it would look like. If we check the results of the process capability levels, there is only one process which has no standardized process in place, which is the process improvement process. All others achieve capability level 3 which means that a standard process has to be established. If this is compared to a SPrL, we can see that this is the basis for a SPrL architecture. Therefore not much has to be done to move from different processes for the different topics of testing to one general testing process. Such a process line architecture is shown in [83]. In [61] the core classes that a process line should contain is displayed. One of those classes is a mandatory feature. This means that for the test process, unit testing is a mandatory feature of the process model. This further includes all of the above mentioned processes from SPICE including integration tests as well. When the approach of [61] is further followed, load testing might be an optional feature which can be chosen when creating a process model, as how to use load tests in the test process. There are also the alternative features, which

replace a mandatory feature with an other similar but more precise feature. This could mean that quality assurance with verification is added instead of quality assurance alone.

A software process line for the test process at Willhaben would have to be very general as only the basic processes for software testing would be general enough for all teams. Such a SPrL would include:

- Planning and control
- Analysis and design
- Implementation and execution
- Evaluating criteria for verification
- Reporting
- Closing of the test activities

These steps are very general and are the basis for every test process. The different techniques used to test the software are unit, integration, system, regression and manual testing, with possibilities to include smoke and load tests. The basic testing techniques are all used in every team, but the usage of additional tests is up to the team. Those testing techniques are work products for the SPrL. As said earlier this is all done at Willhaben, without the declaration of a SPrL, but a clear definition of a SPrL could help to improve the process of testing, as those differ greatly between the different teams. In [61] an example of a SPrL is shown with roles, such as the team lead or the project manager, work products, like the QA plan, and how it is all linked together. As stated by [61] the German V-model XT is an example for a SPrL. This means that many companies that use the V-model are using a SPrL, which is also the case at Willhaben in an adapted form. As the interviews later show nobody at Willhaben knows that it is a SPrL.

When we look at [99] the authors stated that different search queries to get results for SPrLs only gave back 40 publications, which means that in 2014 few papers where written about this topic. We altered the search to add testing as a keyword and changing the time slot to 2000 to 2019. In this range there were 40 results as well, but only one was situated in software development, which was [102]. This means even now few researches are done for using SPrLs in testing. In the more general area a lot more search results were found.

CHAPTER 8

State of Practice Survey

In this chapter the structure of the survey, the approach to the survey and the results we got from this are shown and described. Therefore the survey questions will be summarized and if the information gained from the questions isn't that great, the questions itself will be compressed and all of the questions will be answered in one block. The whole survey with every question and answer will be included in the appendix.

8.1 Survey Design

This section describes the survey design, how the questions were selected, which survey platform was used and how the people for the survey were selected. The time line for the survey is described as well as the composition of the interviews and the selection of the interview participants.

8.1.1 Survey Structure

The survey is separated into six different blocks. Subsection 8.2.1 is about personal information and the occupation of the participant, as well as the years of experience in the position and field of work. In figure 8.1 the structure of the survey is shown.

The second block covers the project and process management, which is described in 8.2.2. This includes the questions if the participant is in an agile team and which software process is used the most in the company. Additionally software testing processes are retrieved as well including if software testing is part of the standard software process.

Subsection 8.2.3 consists of software testing in general and how familiar the participant is with software testing. The general questions outline frameworks that are most popular in the use for software testing and cover unit and integration tests. The satisfaction with the overall software testing is questioned as well.

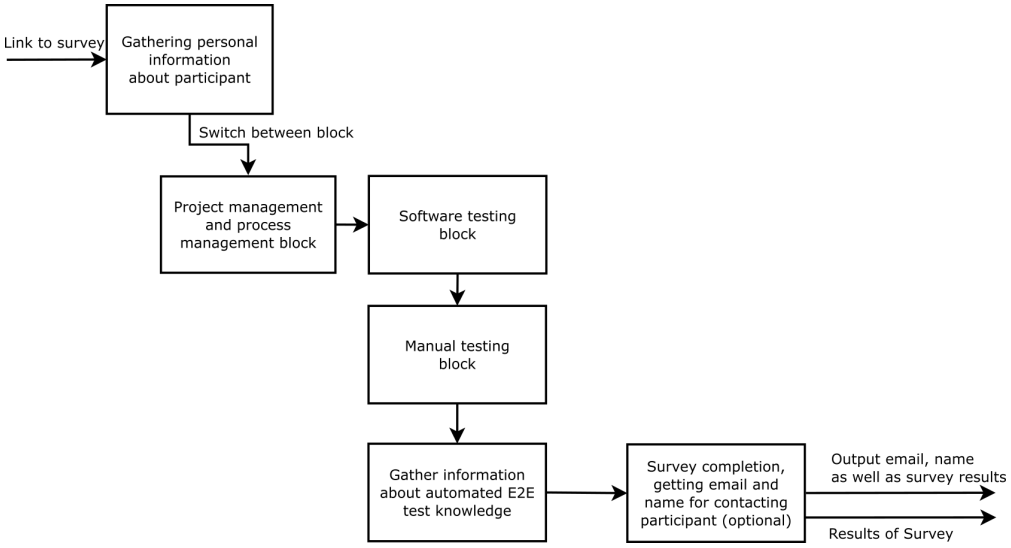

Figure 8.1: Survey design with the 6 different blocks

Block 8.2.4 is about manual testing, how manual testing looks like in the company of the participant, if manual testers are employed and if it is necessary to have manual testers in the respective teams. Additionally the frequency of the execution of the manual tests is asked for as well and which scope is behind it.

The fifth block covers *End-to-End (E2E)* tests and is found here 8.2.5. This includes if E2E tests are even used and which languages/frameworks are used for them. As a second part it is questioned who is responsible for those tests.

The last block 8.2.6 just offers the possibility to give a short feedback on the survey itself, which was used to make the survey better, such as including short comments on what is meant by the different sections, and to write down the name and email address to get the results after the survey was done.

The questions can be split into two sections. One is personal information that has few to do with gaining information, but is necessary to get to know how the participants think of the survey and if they are willing to help more afterwards as well, and the second is about gathering information to use for this work. The questions can additionally be split into mandatory and optional ones. The questions were selected with the experiences made at Willhaben and adding for example the most popular frameworks for a certain question as a possible answer.

The questions asked offer either some answers that have to be selected, most popular answers with the possibility to add an own term for the answer or an open answer that can be anything the participant want to say. Additionally there were some follow-up questions where the participant was asked to clarify why something was done the way it is done.

8.1.2 Platform, People & Timeline of the Survey

The survey was done on Google docs forms which offers free space to conclude a survey. It has a sizeable collection of question templates from which to choose and includes a free analysis for the results to quickly get the results. The finished survey is easily sent to the target audience as the creator can choose to send it either via e-mail or per link. If a new answer for the survey is done, the creator can choose to be notified by e-mail. It is a simple way to create a survey for a large audience and it is configurable to make it more personal and appealing. As stated by [87] the quality of the collection of data should be as good as possible which was guaranteed with Google docs forms as it offers a summary of all answers as well as a single overview over each response. Additionally the data can be exported to an Excel-sheet to perform different tasks on the data.

The participants for the survey are people from Schibsted marketplace companies that work in *Product Management (PM)* or *Product Development (PD)*. This includes software engineers, team leads, software testers, people from operations and designers that are working in PM. The test and initial group consisted of people that work for Willhaben, which is the work place of the author of this work. The initial group was selected to check the survey for errors and give inputs on how the question should be phrased.

The survey was open for five months from Mai until October 2018, but the first draw of the survey was done earlier in January of 2018. The responses were expected to be in the range of 50 to 100, as the quantity of people in the different teams and companies were about 800 to 1000. The return rate is low, as many people didn't respond to the survey. The first draw was sent only to a few people that work at Willhaben to get their initial feedback and to get a feeling on how long the survey is. This response size represents about 6-8% of the people which work in the different companies. Those two factors also determine if a survey design is good or bad as said in [87]. As this percentage isn't that high the possibility of false answers to a company is higher when only one participant was from a company.

8.2 Survey Results

In this section the survey results are summarized and graphics and tables are shown of how the participants answered the different questions. Every block of the survey is described.

8.2.1 Personal Information

The first block of questions was about personal information, to get an idea of the person. There were three questions regarding the company, the department and the position of the person. In figure 8.2 the amount of people that took the survey can be observed. As expected most of the answers came from Willhaben, because the author works at Willhaben and has a personal connection to the people there. The information here is summarized, as the survey tool didn't compress the answers to such a degree that it was

well readable. This is done for every open question that was asked. This is necessary especially for the open questions as the answers couldn't be summarized if the answer was slightly phrased differently.

As can be seen in the survey-answers a lot of different people from different departments and jobs answered. The main departments were IT and PD. This was expected as the survey was mainly sent to the tech-department and to PD. The departments are quite similar for all companies and include either PD, *Product Management (PM)* or the *Quality Assurance (QA)* department.

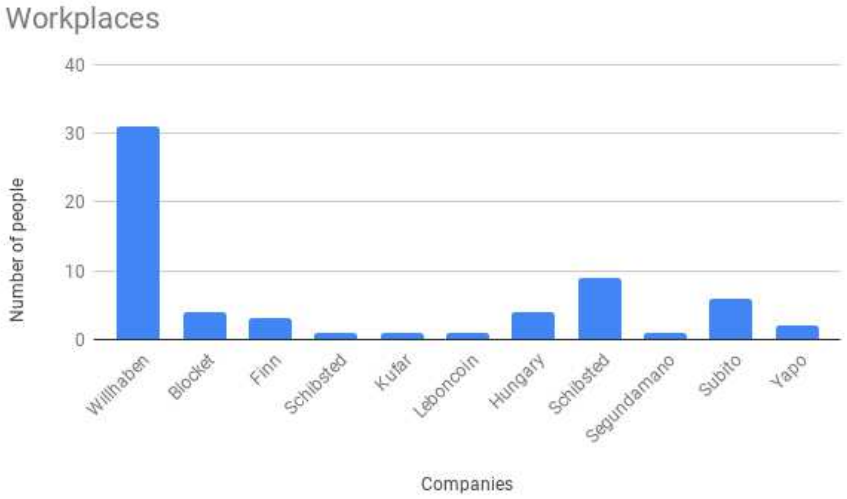

Figure 8.2: Workplaces of the participants

In figure 8.3 the experience in the current position and the overall experience in the field of work in which the participants work are shown. As can be seen the majority has been working in the same position for 3-5 years, but the majority have worked more than 10 years in the field. This indicates that the persons have either worked for other companies as well or they changed to an other position inside the company.

8.2.2 Project and Process management

In the second block of the survey the aspects of project and process management were highlighted, especially if the teams have a specific development process or included software testing in the planning phase of the project.

In figure 8.4 the development process that is currently in place for the software can be observed. The majority of the teams are considered agile or as a form of agile, but not every team, as some use a traditional approach to software development with a waterfall or a V-model. There are some people that answered that they are working in an agile

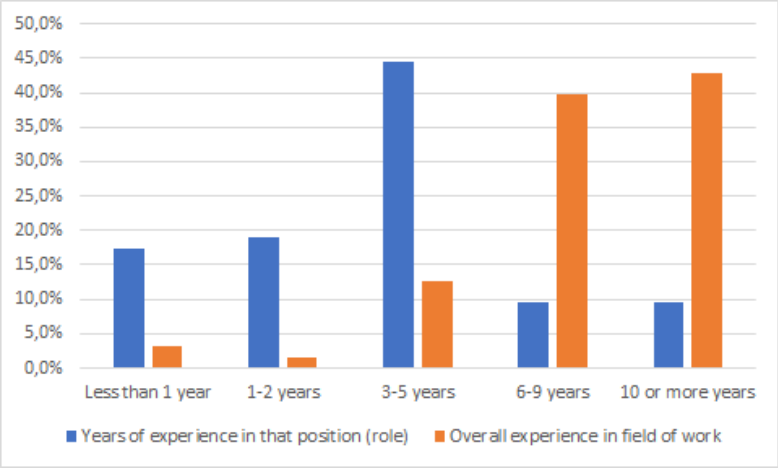

Figure 8.3: Experience in the current position and the overall experience in the field of work

team even though the model for the team isn't done completely right or was changed to fit the needs of the team. As can be seen in figure 8.4 only 7,9% of the participants are working in a team which uses the traditional process.

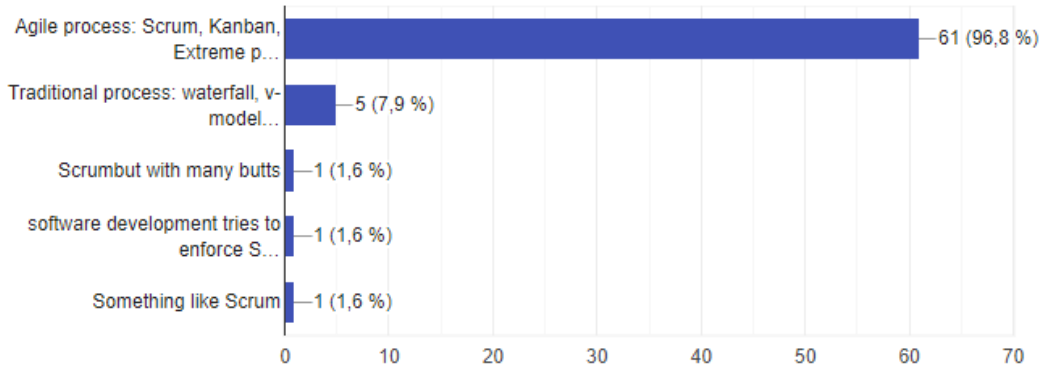

Figure 8.4: The software development process that is used in the different teams

Question eight is about the most used software development process in the teams, to get a more specific answer than in question seven, as there the big difference was if the participants team use traditional or agile process management. In figure 8.5 the most used software development processes are shown. Agile as an answer has to be understood as the general term for every agile process that is known by the participant. Scrum and Kanban are the two most used ones, with Scrum as the clear dominator. The answers were summarized, as some people said they have Scrum in place, but as said before for

Software Development Process	Number of people	Percentage
Scrum with deviations of Scrum	35	65%
Kanban with deviations of Kanban	7	13%
Agile	5	9,3%
Mix of different	2	3,7%
Devops	1	1,8%
Implementation and verification	1	1,8%
Jira	1	1,8%
Git flow	1	1,8%
Waterfall modell	1	1,8%
	54	100%

Table 8.1: Table of Q8 summarized

question seven, it'sn't Scrum as it is defined in theory.

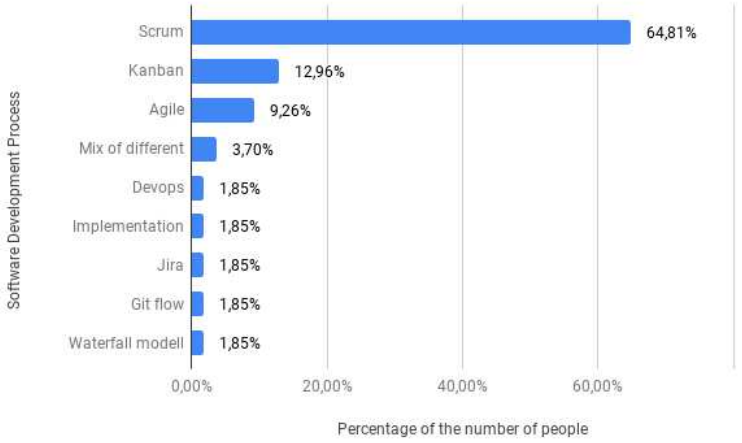

Figure 8.5: The most used software development process of the participant

In table 8.1 the amount of people that answered is 54. This isn't the amount of people that conducted the survey, but as said in 8.1.1 this is due to the fact that not every question is mandatory. Here the results are shown in absolute numbers as in comparison to figure 8.5 where the percentage is used. The survey result looks different as well, as this question allows an open answer and therefore the answers have to be summarized as the tool doesn't add up all spellings of the same word. This has to be done by hand.

In figure 8.6 the percentage of people that have software testing as a part of their project plan is shown. Only one quarter of the participants said that the either don't have or don't know if software testing is part of their project planning cycle. So the majority has software testing very early on in their projects. Only one person didn't answer this

question.

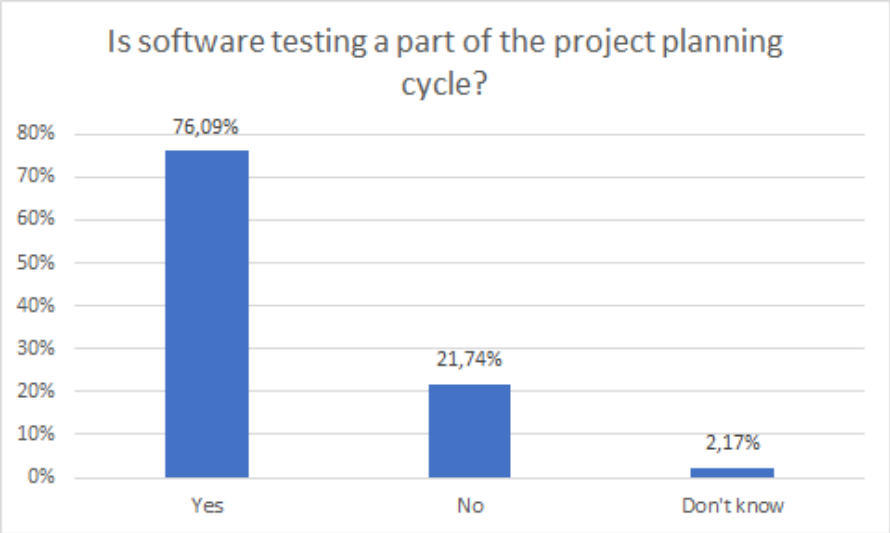

Figure 8.6: Software testing part of the project planning cycle

The next question regards the different software tests that are used in the software development lifecycle, regardless if software testing is part of the project plan, as software tests can be developed and used after the general development is done. This does also include manual software tests, which have to be written down or at least have to be planned before the execution of them. Every participant answered this question and as it can be seen in the figures 8.7, 8.8, 8.9, 8.10, 8.11 not everybody knows if software tests are even used for the projects that are developed or even worse they don't use even the most basic test type which is Unit testing.

Which software tests types are part of the software development life cycle?	Unit tests	Integration tests	E2E tests	Regression tests	Manual tests
In every project	42 - 66,7%	21 - 33,3%	9 - 14,4%	21 - 33,3%	30 - 47,6%
Often	12 - 19,0%	27 - 42,9%	20 - 31,7%	17 - 27,0%	18 - 28,6%
Seldom	7 - 11,1%	9 - 14,3%	14 - 22,2%	14 - 22,2%	7 - 11,1%
Never	1 - 1,6%	2 - 3,2%	7 - 11,1%	2 - 3,2%	5 - 7,9%
Don't know	1 - 1,6%	4 - 6,3%	13 - 20,6%	9 - 14,3%	3 - 4,8%

Table 8.2: Table for question 10 of the survey

In table 8.2 the different answers for question 10 are summarized. Many people answered

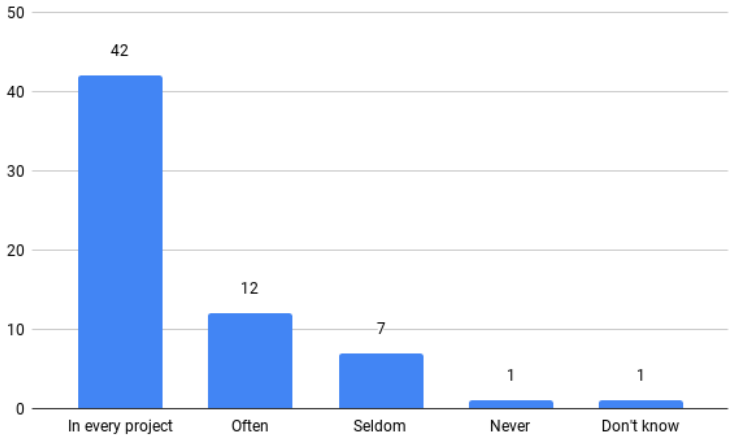

Figure 8.7: Unit testing as part of the development life cycle

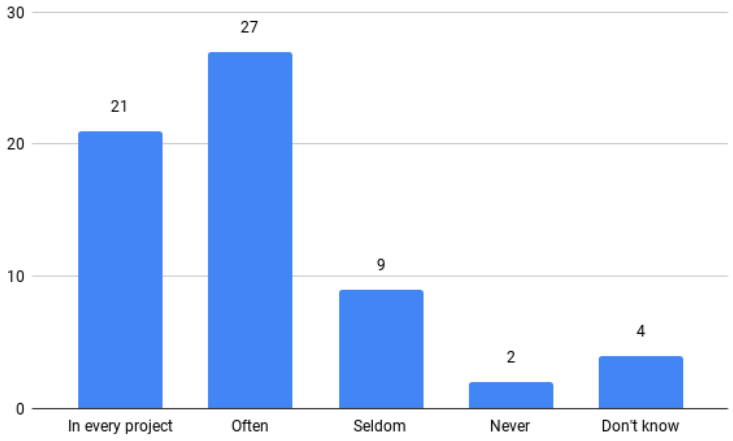

Figure 8.8: Integration testing as part of the development life cycle

that they don't know if E2E or regression tests are used for the projects they are working on. This indicates that it'sn't communicated if and how tests are done in the respective companies and that communication isn't done correctly. PD should know which tests are executed and how those are done. Every figure for the data of the table has to corresponding label.

Question 11 was about the estimated testing effort that is done when a new feature or a new project is implemented and developed. This means that the testing effort is compared with the effort of the software development in total and how much time is spent for the tests when new features in the software are developed. The answers can be

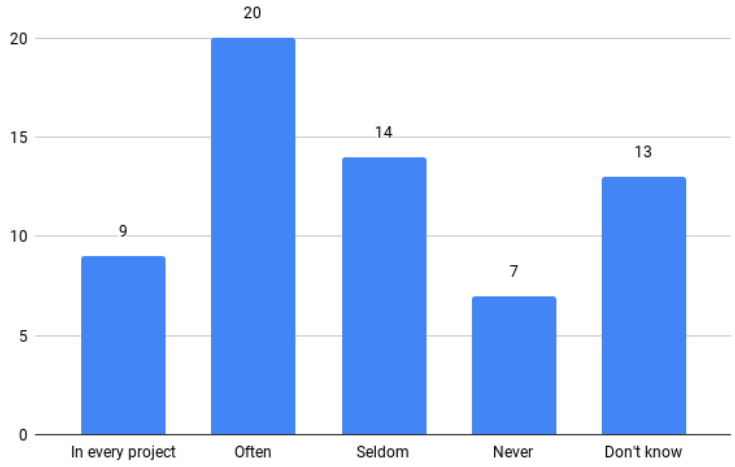

Figure 8.9: E2E testing as part of the development life cycle

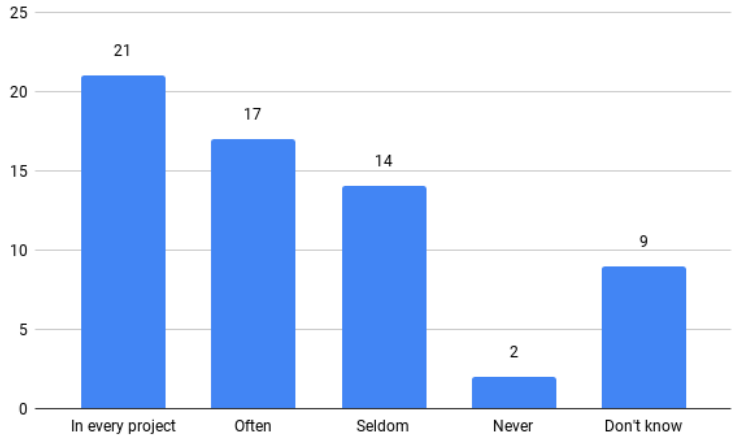

Figure 8.10: Regression testing as part of the development life cycle

seen in 8.12. Most of the people said that the testing effort is around 25 to 50 percent. This will probably differ if a feature is bigger or smaller, but it gives a good indication on how much time is spent on testing. A lot of people doesn't spend a lot of time for writing tests and testing in general. Sometimes it is hard to determine how much time is spent for testing and some people don't develop features and components at all, so they can't know how much time is spent for testing. This is shown by the number of people that selected "Don't know" as their answer.

In figure 8.13 the main type of tests and the focus on which tests are used is shown. The main type is Unit testing but at 36,4% manual testing isn't far behind, which isn't a good

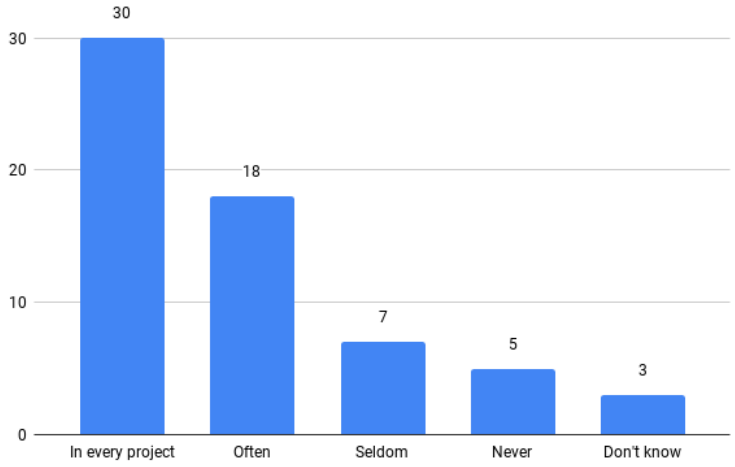

Figure 8.11: Manual testing as part of the development life cycle

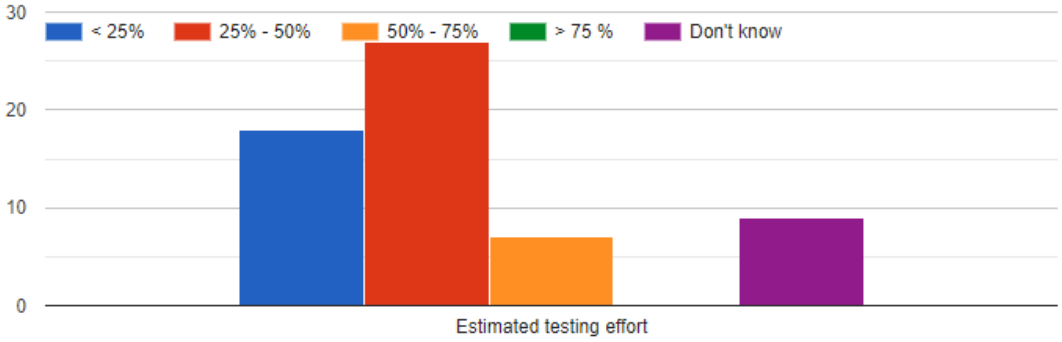

Figure 8.12: The estimated testing effort in comparison to the overall development effort

sign as manual tests should be used far less. The also shows that integration and E2E tests are used more or less the same in the respective companies. When comparing this to the test pyramid that was presented in chapter 1 the share should be a lot smaller.

8.2.3 Software Testing

Software testing in general is the topic of the third block and the answers were quite surprising for the most part, as many well known practices are still not in place in some of the companies, or aren't fully developed to a degree that could have been expected. This is often explained with legacy systems which are hard to test and have weird

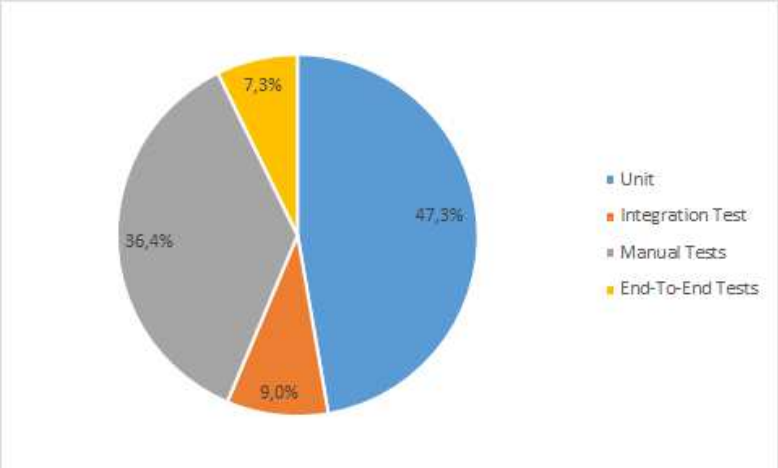

Figure 8.13: The main type of tests that are used

behaviors when touched. This doesn't mean that there aren't companies where testing is sophisticated enough to keep up a high standard of the software, but the majority lack some aspects that are standard in most researches.

In the first question of the software testing block the familiarity of the participants with software testing in general is checked. As can be seen in figure 8.14 around 75% said that they are actively participating in the software testing process. This includes of course the software testers as they have to be familiar with software testing.

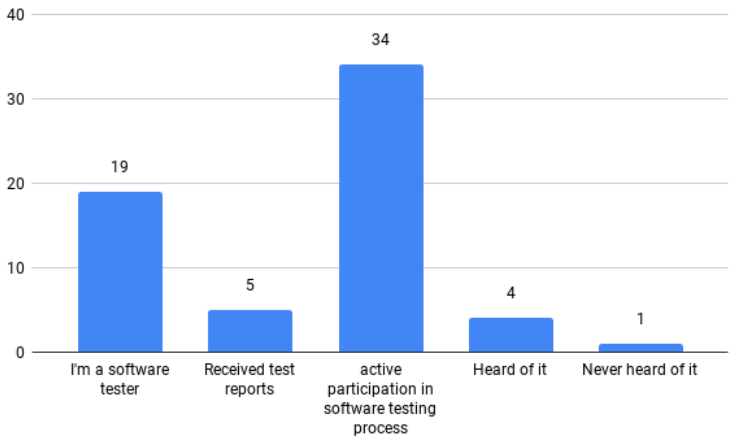

Figure 8.14: Familiarity of participants with software testing

Only one person said that they have no familiarity with software testing, but there are

also four persons that have just heard of it. This is the minority and not every person is a developer or a software tester so this can happen.

Question number 14 was about the programming languages used in the respective companies and projects. There were five possible answers and the most used programming languages were listed to give the participants some choices for their answers. Additionally there was an open text field, where participants could put other languages which they use or have used. In table 8.16 the answers are counted to get an idea how the language usage looks like. In figure 8.15 the percentage of how often a language is used in a project is shown.

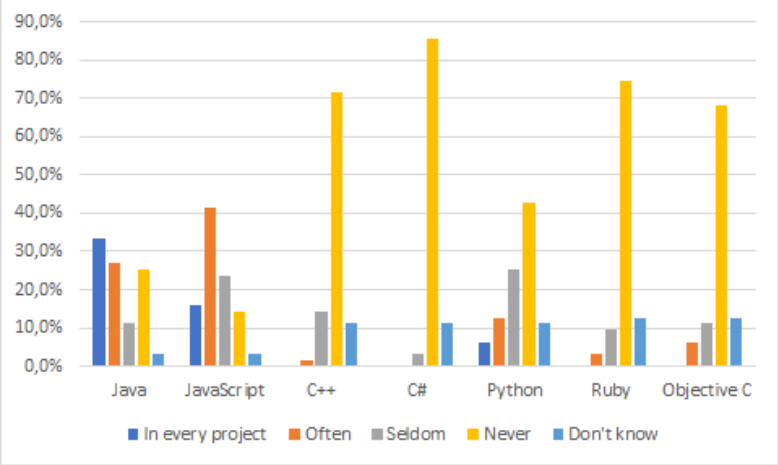

Figure 8.15: Languages used in projects

	Java	JavaScript	C++	C#	Python	Ruby	Objective C
In every project	21 – 33,3%	10 – 16,1%	0 – 0%	0 – 0%	4 – 6,5%	0 – 0%	0 – 0%
Often	17 – 27,0%	27 – 41,9%	2 – 1,6%	0 – 0%	9 – 12,9%	2 – 3,2%	4 – 6,5%
Seldom	7 – 11,1%	15 – 24,2%	9 – 14,5%	2 – 3,2%	16 – 25,8%	6 – 9,5%	8 – 11,2%
Never	16 – 25,4%	9 – 14,6%	45 – 72,6%	54 – 85,7%	27 – 43,5%	47 – 74,6%	43 – 69,4%
Don't know	2 – 3,2%	2 – 3,2%	7 – 11,3%	7 – 11,1%	7 – 11,3%	8 – 12,7%	8 – 12,9%
	63	62	62	63	62	63	62

Figure 8.16: Languages used in projects

In table 8.3 all of the previously mentioned programming languages, that were written down in the free text area, that are used as well in projects, are listed. There are some that are well known, such as PHP, Golang (Go), Swift, which is often used for applications on iOS, or Kotlin. Those last two are newer in comparison to the other ones in the list and as can be found at [122, 123]. Plpgsql is a procedural language for PostgreSQL and is only used by one team.

PHP	Golang	C	Swift	Kotlin	plpgsql	Haskell	Scala	Typescript
8	13	5	6	7	1	1	3	2

Table 8.3: Other programming languages that are used

As a follow up on question 14 the participants were asked to list any other programming languages that they use for unit or integration tests. The most mentioned language here was Groovy. Another popular choice for a language was Go again.

The next question was about different frameworks that are used for unit and/or integration tests. As like before in question 14 the participants had some choices of frameworks, but could list others as well. The most used framework is JUnit by far, as it is used at least often by 32 people of the 50 people that answered this question. There are also some participants that use TestNG, PyUnit or other lesser used frameworks, as can be seen in figure 8.17 and in table 8.4. The table and the figure don't show all frameworks that were listed, as some weren't used at all, such as Randoop. Those never used frameworks are lesser known and typically for programming languages that aren't used in the companies that participated in the survey.

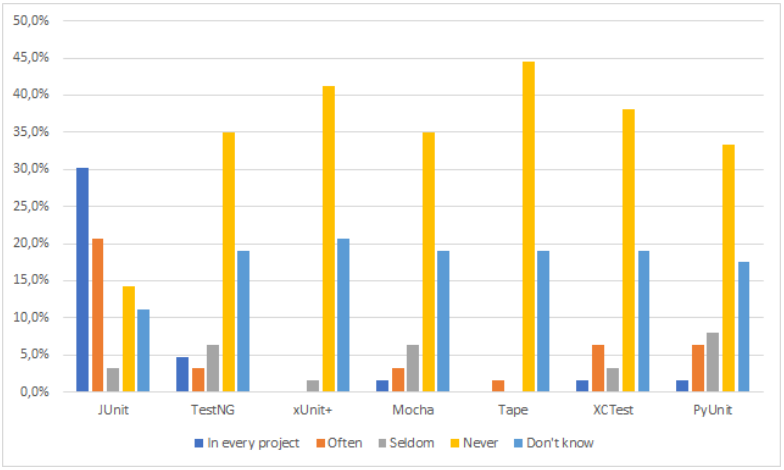

Figure 8.17: Different frameworks used for developing unit and integration tests

As the last question of this section, the participants were asked how satisfied they are with the amount of testing that is done in their companies. There are 2 persons that say that they are fully satisfied with the testing, but there are also only 2 that are completely dissatisfied. The majority is satisfied to a certain degree, but some small things are missing for them. This becomes clear when the statements are read that were given as an explanation on their answer.

In figure 8.18 the satisfaction of the participants with the amount of testing can be seen. The scale reaches from 1 to 6 which corresponds to poorly satisfied to fully satisfied.

	JUnit	TestNG	xUnit+	Mocha	Tape	XCTest	PyUnit
In every project	19	3	0	1	0	1	1
Often	13	2	0	2	1	4	4
Seldom	2	4	1	4	0	2	5
Never	9	22	26	22	28	24	21
Don't know	7	12	13	12	12	12	11
	50	43	40	41	41	43	42

Table 8.4: Frameworks used for developing unit and integration tests

	Junit	TestNG	xUnit+	Mocha	Tape	XCTest	PyUnit
In every project	38,0%	7,0%	0,0%	2,4%	0,0%	2,3%	2,4%
Often	26,0%	4,7%	0,0%	4,9%	2,4%	9,3%	9,5%
Seldom	4,0%	9,3%	2,5%	9,8%	0,0%	4,7%	11,9%
Never	18,0%	51,2%	65,0%	53,7%	68,3%	55,8%	50,0%
Don't know	14,0%	27,9%	32,5%	29,3%	29,3%	27,9%	26,2%

Table 8.5: Percentages of frameworks used for developing unit and integration tests

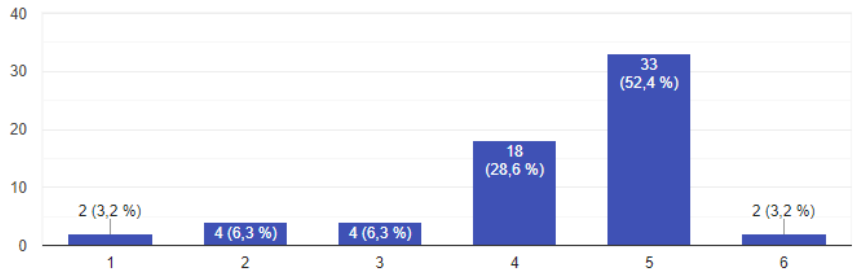

Figure 8.18: The satisfaction of the participants with the amount of testing that is done

The comments to this answer range from everything from is fine and there can't be enough testing but it's on a good level to there are no tests at all, there is legacy code which is poorly tested or there are too many manual tests and we should get rid of them. This of course depends strongly on the company and as said before over 80% are satisfied with the amount of testing, but it's still an indicator that there are things that could be done better.

8.2.4 Manual Testing

The fourth block is about manual testing and its consequences on the company. Most of the participants were agreeing with the general persuasion of manual testing and with the research that is done in this area.

With the first question of this block the participants were asked if the company they work for employs explicit manual software tester. As can be seen in figure 8.19 about 2/3 of the companies do have software testers, which are doing manual testing.

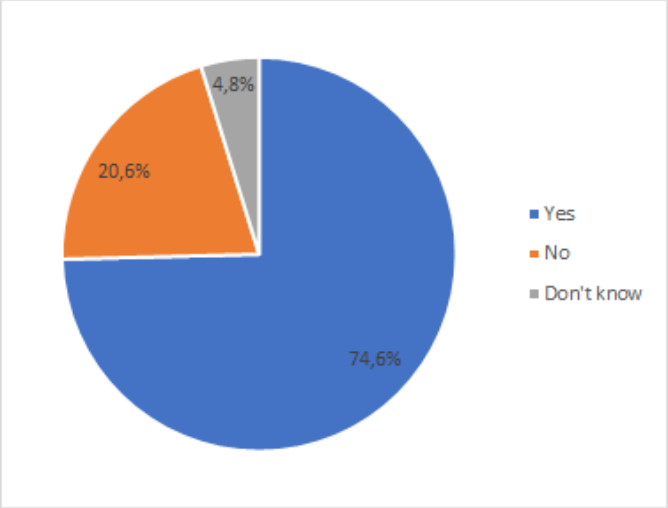

Figure 8.19: Company employs explicitly manual testers

This question is related to the other questions in this block, but every question can be answered even when no manual testers are employed.

In question 18 the composition of the development teams is questioned regarding if they have manual testers in those teams. This is shown in figure 8.20. Again the majority has included the manual testers in their teams, which indicates that the testers are important for the company and have some kind of influence on the teams.

The manual testers are in case of Willhaben always a part of a certain team, which will be taken into account in the discussion. The team structures of other companies are not known, but are asked in the test expert interviews.

Figure 8.20 is also supported by the next question which consists of two separate parts, as it had to be done in two questions in the survey. The question answered if manual testing is necessary in the various projects. As before over 57% answered with yes. As can be seen in figure 8.21 27% stated that it depends on some factors. These factors were included from the participants in the follow up question.

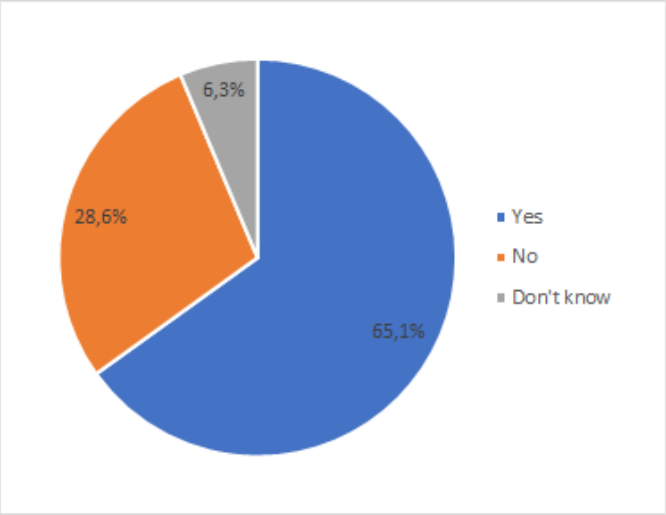

Figure 8.20: Manual testers are part of development teams

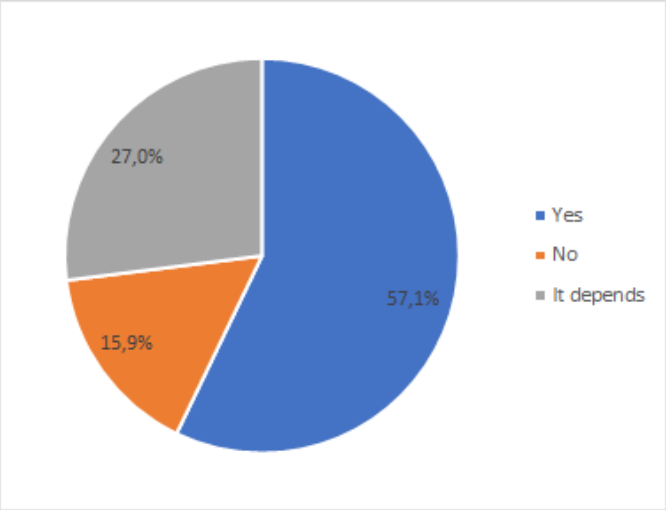

Figure 8.21: Manual testing required in various projects

Some of them stated that manual testing is necessary especially for edge cases others said that projects with a *Graphical User Interface (GUI)* have a higher demand in manual testing but they could be spared if the GUI is absent. Flaky tests are also a reason for manual testers to be used in projects, as well as mobile application, as those have few possibilities to rely heavily on automated tests. Visual effects such as advertisements are also mentioned as hard to be tested with software tests, as those can't detect visual bugs. Another point that was noted was that a manual has a feel for the application that is

tested and can say if it feels right. This and the fact that testers do think differently than developers are interesting points that were mentioned.

On the other hand the minority which said that there is no need for manual testing in projects says that good software should be software tested in such a way that it is sufficient. Some have the approach that everything can be automated and should be if just enough resources are spent on it. Some say that manual testing was declared inappropriate by the organisation and therefore they were stopped. In addition it was mentioned that *Continuous Integration (CI)* or *Continuous Deployment (CD)* can help with fixing bugs faster that are in production when automated tests help to secure the main parts of the product. When the bug is fixed a new automated test is added and catches the reappearance of this bug.

Question 21 is about the amount of manual testing and if it should be increased. This was denied by 71,4% of the participants, which as expected, as manual testing as shown in section 1.2 and 2.2, where the testing pyramids are shown, are considered slow and cost intensive. This shown in figure 8.22.

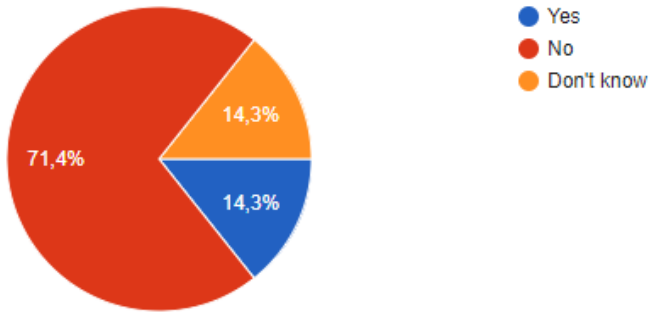

Figure 8.22: Should be the amount of manual testing be increased?

The participants were asked to explain their answer and most of them said that they rather have the amount of testing that is currently done, but avoid more manual testing as it slows down the whole release process. It is stated that manual testing doesn't scale in comparison to automated software testing. Most said that they would prefer it if the amount of automated tests is increased and that the testers write more automated tests themselves.

In figure 8.23 the results of the next question are shown. The participants were are asked how often the manual tests are executed and was a surprising was that 19% or 12 persons said that the tests are never executed. This of course is related to questions 17 and 18, where 13 people said that they have no explicit manual software testers and 18 people stated that manual testers are not part of development teams. The majority of 36,5% still claimed that the tests are executed for every feature change. This is an indicator that some companies still rely heavily on manual tests.

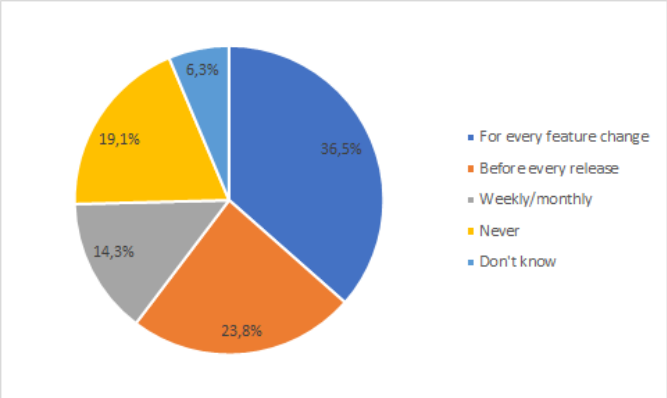

Figure 8.23: How often are the manual tests executed?

The scope of manual testing was the next question, which can be seen in figure 8.24. There were 5 possible answers, those are changes, new features, according to the test plan, current release and don't know. The 5 option of the one that was chosen the most, with 27,9%, which is quite high when considering that the question wasn't mandatory and most participants are developers or testers. The other answers were chosen evenly distributed except for the third one.

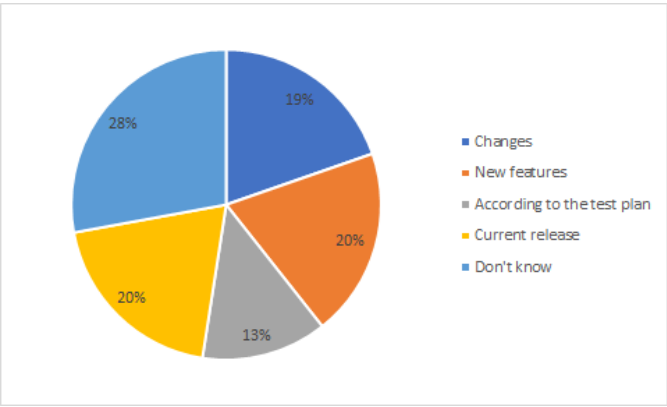

Figure 8.24: The scope of manual testing

The final question of this block was about the management of the tests, if they, for example, are done according a test plan, test reports or with test recommendation. This was an open text question so the answers were quite different and only 27 persons even answered it. The most used management tool is JIRA which was mentioned five times in the answers. JIRA is a issue and project tracking software, where PM can put specification and feature tickets and testers can write down issues. Other answers said that a test plan is used or a checklist with the main features of the new release. Test

hints according to the tickets is also used as a method to keep up the software testing. Every answer can be found in the survey section that is included at the end.

8.2.5 Automated End-To-End Tests

The fifth and penultimate block tries to find out how automated E2E tests are perceived and if they are used in the respective companies. Those questions are important for the overall work, as other researches only briefly touched the inclusion of E2E tests in a development process and in a SPrL.

At first the general usage of E2E or functional tests, which is the hypernym and includes E2E tests within it, was deducted and in figure 8.25 it can be seen that the majority uses both. This indicates that there are probably functional tests used that are E2E tests as well. 23,8% said that they don't know if E2E or functional tests are used, which corresponds to 15 people, which is a high number considering the participants that were used. This has to be looked into if people from the same company gave different answers to this question as this would point at poor communication inside the company.

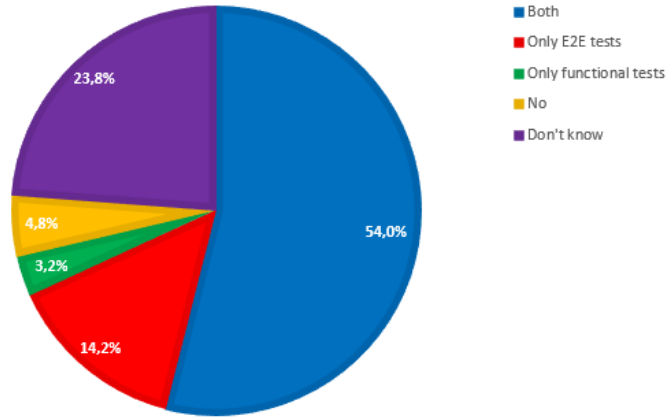

Figure 8.25: Usage of E2E or functional tests in various companies

Are E2E or functional tests used in your company?	Nr. of people
Both	34
Only E2E tests	9
Only functional tests	2
No	3
Don't know	15

Table 8.6: Table for Q25, with the number of people for each answer

E2E tests can be written in different languages and this was asked as the next question. The participants had seven programming languages to choose from and had a text field to write down other languages they use. In figure 8.26 the usage can be seen. In table 8.27 the answers are shown as well, it can be seen that not every body answered the question, or incomplete, as someone selected one answer but left another one out. C++ is the language nobody uses for testing, at least not in the companies that are part of Schibsted.

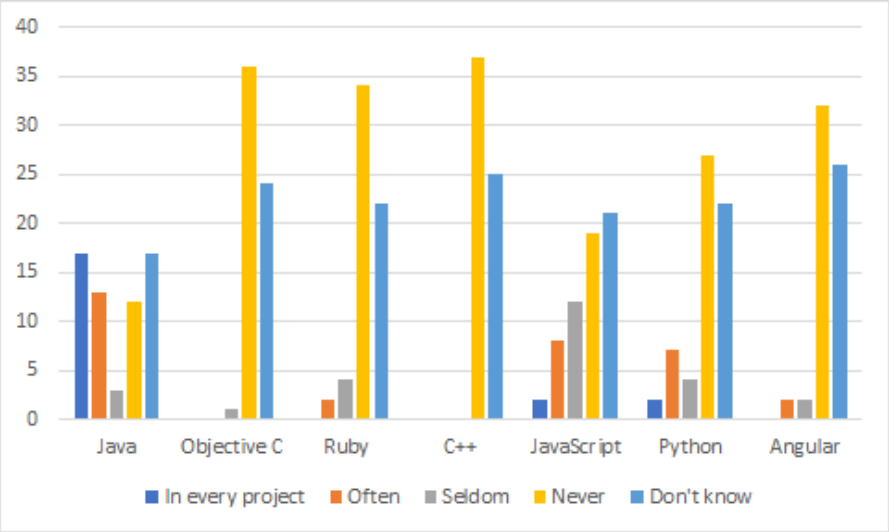

Figure 8.26: Languages used for developing E2E tests

This doesn't mean that C++ can't be used for testing, but it is just not used in the sample companies.

	Java	Objective C	Ruby	JavaScript	Python	Angular
In every project	17 – 27,4%	0 – 0%	0 – 0%	2 – 3,2%	2 – 3,2%	0 – 0%
Often	13 – 21,0%	0 – 0%	2 – 3,2%	8 – 12,9%	7 – 11,3%	2 – 3,2%
Seldom	3 – 4,8%	1 – 1,6%	4 – 6,5%	12 – 19,4%	4 – 6,5%	2 – 3,2%
Never	12 – 19,4%	36 – 59,0%	34 – 54,8%	19 – 30,6%	27 – 43,5%	32 – 51,6%
Don't know	17 – 27,4%	24 – 39,3%	22 – 35,5%	21 – 33,9%	22 – 35,5%	26 – 42%
	62	61	62	62	62	62

Figure 8.27: Table for Q26: Which language are you using for these tests?

Other languages that were mentioned were Go, PHP, Scala, Groovy, Kotlin and Swift in descending order. Those other languages were only mentioned by 8 people, so most people use the pre-selected languages for their tests.

In question 27, the responsibility of writing the tests was settled. The two most selected answers were that developers and quality assurance are responsible for it. In figure 8.28 it can be seen that 8 persons don't know who has the responsibility to write E2E tests, which can be unclear for various reasons but the responsibility should be settled.

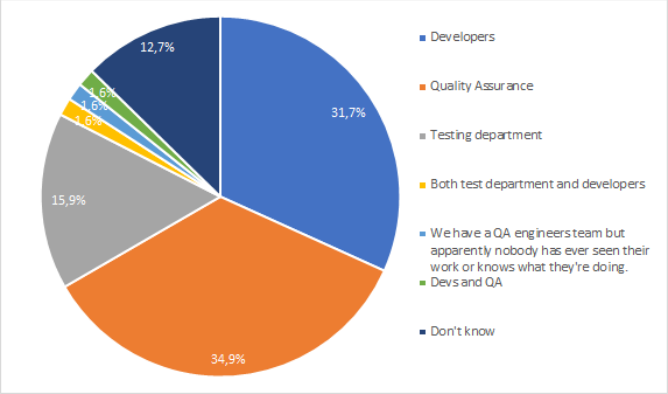

Figure 8.28: Who has the responsibility to write the E2E-tests?

As the last question of the block the used frameworks were of interest. Again a range of different popular frameworks were given to be selected and an open text field to get other frameworks as well. The most used framework was Selenium, which can be used as a basis for many other frameworks. In figure 8.29 the different frameworks that are used for the E2E tests are shown. In table 8.30 the usages of the different frameworks is shown in absolute numbers.

This includes Selenium, Appium, Calabash and Gherkin with Cucumber. Those are the main frameworks that are used at Willhaben, but as before, the participants were asked to add any additional frameworks they have used before. This was done by 12 persons who stated something different than the 4 mentioned earlier. Protractor wasn't used by anybody and therefore we didn't include the answers on this framework in table 8.30. Every answer has a different number of answer, this is because of the people just ticking the first answer which is enough to be accepted as a valid answer in Google forms.

8.2.6 Survey Completion

The last questions are about the survey, if something could have been improved and if we could reach out to the persons and get more information about their testing experience. Some of these recommendations were small improvements to a specific question or rephrasing a question as it wasn't clear what was meant by it. Those little improvements were done immediately as they affected the survey only in a positive way. Other requests or recommendations were considered but not changed, as it would have meant to delete a whole question or add a new one. As said before the second part was to ask for permission

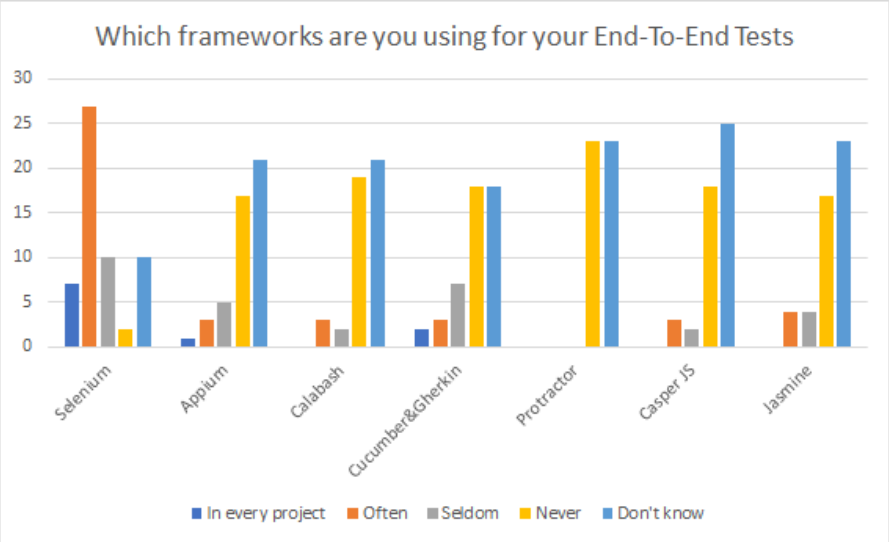

Figure 8.29: Frameworks used for developing E2E tests

	Selenium	Appium	Calabash	Cucumber & Gherkin	Casper JS	Jasmine
In every project	7 - 12,5%	1 - 2,1%	0 - 0%	2 - 4,2%	0 - 0%	0 - 0%
Often	27 - 48,2%	3 - 6,4%	3 - 6,7%	3 - 6,3%	3 - 6,3%	4 - 8,3%
Seldom	10 - 17,9%	5 - 10,6%	2 - 4,4%	7 - 14,6%	2 - 4,2%	4 - 8,3%
Never	2 - 3,6%	17 - 36,2%	19 - 42,2%	18 - 37,5%	18 - 37,5%	17 - 35,4%
Don't know	10 - 17,9%	21 - 44,7%	21 - 46,7%	18 - 37,5%	25 - 52,1%	23 - 47,9%
	56	47	45	48	48	48

Figure 8.30: Table for Q28

to include the person in an interview and if the person would like the results of the survey.

CHAPTER 9

Expert Interviews

In this chapter the test expert interviews are described, analysed and summarized. In the first section the structure of the interviews is explained and with which method the interviews were conducted. After that the interviews are analysed and summarized and the results are presented.

9.1 Expert Interviews Structure

The test expert interviews were done after the survey was finished and the people were chosen that answered the question if they wanted to be contacted for further questions with yes. From this group of potential interview partners the selected ones either are in interesting positions or had something to do with test automation or testing. Not everybody contacted responded and because of that only 10 people were selected to be interview partners. Some of the interviews were conducted in the time between the survey refinement before it was opened to everyone in the research group. This was done because it was necessary to see how the interviews had to be done when different answers were given.

The test expert interviews aren't all the same, as the interview question had to be changed for the participant and the answer he/she gave in the survey. For example a person that is working in an agile team has other insights and answers than a person in a traditional team. Additionally, persons of different levels were asked to do the interviews to get a diverse set of answers.

As stated in [85, 86] an interview is an information exchange between the interviewer and the interviewee. There are different interview forms which all serve different purposes. Interviews are, as stated by [86], conversations that have a structure and a purpose. For this work the purpose is to get information about the software test process and a deep understanding about how it is used in the different companies, with a focus at Willhaben,

as it will be used as a comparing element. As said before the interview is changed upon the answers that are given, which is one way to use the qualitative interview, as it is also called an unstructured or non-standardized interview [86]. The answers also determine if further questioning is needed to get more information from a person, or if the answer is satisfying enough for the purpose of the interview. As stated in [89] the interview that is used in this work can be qualified as a semi-structured interview, where some predetermined open-ended questions are used with follow-up questions that emerge through the dialogue between the interviewer and the interviewee.

The three main questions every participant was asked were:

- How does the current test process look like in your company?
- Do you think the current test process could be improved? And if yes, how can this be achieved?
- Do you think that a Software Process Line in regards to testing could support the test process?

The third question had to be explained to all participants as *Software Process Line (SPrL)* was not a familiar term. Some participants had already heard about it, but no one knew exactly how to use SPrLs in testing.

As said before the questions that followed the three main questions, were individual and therefore the length of the interview depended on the answers as well. There was no time restraint on the interviews, as the only factor was that the questions were answered thoroughly. In [89] the authors state that an semi-structured interview last between 30 minutes to several hours. As this interview was based upon the survey that was done earlier, the duration of the interview was between 15 minutes and 60 minutes as only three main questions were answered. Therefore it's a short interview, but nonetheless provide data for this work.

This interview, as stated by [86, 89], is a form of a production of knowledge interview, therefore produce knowledge as an outcome of the conversation between the interviewer and the interviewee.

The interview will not be transcribed which would be the case if we followed the exact procedure in [86], but the analysis, verification and reporting will be done. This is a measure to reduce the possibility that confidential information is introduced in this work. Additionally, it doesn't bring any value to this work, as the essential information is summarized and analysed.

In [85] the anonymization of the data is described, as this has to be considered for securing the personal interests of the participants. Because of the summary of the results the interview participants aren't named, as the different opinions on the matter will be compared without saying who said something.

Every participant was asked before the interview if voice recording was accepted. If they had refused, the interview wouldn't have been conducted, as the recordings were needed to compare the different answers. Getting consent from the participants is an important factor, as data protection has to be done and personal data is collected [85].

The interview was conducted with either one or two participants at the same time and was not a group interview [89]. In this case it was possible to get the honest opinion from every participant, which could have been concealed if more than one person had been questioned at once.

9.2 Results and Analysis of Expert Interviews

In this part the answers of the interview partners are summarized and analyzed. The interviews will not be transcribed, as some of the information that was given is confidential and can't be published. The interviewees were from different countries, this includes Austria, Hungary, Italy and Norway. As stated before there were only three main questions, as a lot was covered with the survey.

The first question was about the current test process that is in place in the different companies. Here the participants answered with a wide variety of answers, as the testing processes are different in each of the teams, especially regarding the current status of the marketplace. If the classifieds web page belongs to the established marketplaces as said in section 4 the test process is more mature compared to the other companies. This is the case for Austria, Italy and Norway, the only exception being Hungary, which is showcased with the answers given by the interviewee. The test process in Hungary is more manual than any other and relies heavily on the manual testers, with little to no help from the developers and test automation. This is different in the other companies where manual testing isn't the number one priority any more or is completely removed as in Norway, which is the other extreme. The test automation in Norway is sophisticated enough that the developers are confident that they don't break anything major if code is pushed to production. They use continuous deployment to push everything that passes their tests immediately to their customers. These customers are then their manual testers which report back any bugs they find or if the whole page breaks down. In the later case the previous version is rolled out and the latest changes are checked for errors that weren't found by the tests. The tests are then adapted to provide better coverage. In Italy and Austria the test process is somewhat in the middle, with manual and automated testing in place.

The test process for the mobile applications, iOS and Android, are more manually tested in Austria, where the test automation isn't as far as in other projects. This causes problems, as the manual testers have to delay a release more often than elsewhere. This is done better in Italy, where the test automation for the apps are further developed and they can rely on the test results so that the manual testing effort is reduced to a minimum. In Norway there are no manual testers at all, which causes problems as the continuous deployment approach fails more often, as the manual verification is missing

for the released application. As a re-roll-out can't be done as quickly as on the self-hosted web-page, because of the restrictions of the Google and iOS stores. Additionally, automated *End-to-End (E2E)* tests aren't used either, which hurts the developers, as they have to do the testing on their own. In Hungary there are no app tests done with E2E tests and only manual verification is done. This slows done the deployments and releases and should be avoided. The best way to do it would be as shown in Italy, where the automation of the tests is far above the other companies. Here all the other companies should adapt and introduce either a small amount of manual testers to verify the developed features or strengthen the automation to such a point that it is reliable.

As the participant said the test process in their respective countries could be improved which was the second question in the interview. Most of them stated that the current test process has its strengths but is weak in other sections. The test process as it is at Willhaben has to be faster according to the team lead of *Quality Assurance (QA)*. This doesn't mean that the testers have to test faster, but to get away from one release per week to more than one and being more stable even though more releases are done. The release cycle which is dependent on the test process, because the tester has to accept the code that is delivered. This has to be measured much more strongly and how long it takes from feature to being deploy ready. Additionally, the test automation should be improved to a certain degree, to relief the manual testers from repetitive work. The developers have to do more automation as well, this was improved but there could be done more. Another approach is that the manual testers and their resources should be handled differently, so that they can be used as effectively as possible. The approach of *Test Driven Development (TDD)* or even further *Behaviour Driven Development (BDD)* and the use of it as a development practice was a follow-up question for some of the participants.

The tests currently run on the development environments, which is a problem for the test process, as false positives often occur and this could be changed when a dedicated test environment is started and after the execution of the tests this instance is killed again. Load and system tests could then be used as well in the test process, which isn't the case right now.

Another point was that the testers don't come into contact with the feature before it's being developed. This could be improved as well, as they could be included in the planning phase as well. This is only partly done in the app development, where the testers and developers are included in the feature specification process.

The participants also think that the *User Interface (UI)* automation could be improved, especially with more personnel working on creating and writing feature tickets. This is useful if a tester finds a bug and writes a bug ticket he/she could additionally identify if there is a need for an automation ticket.

The third question was about the introduction of an SPrL and if it could support the current test process. The concept of an SPrL had to be explained to every single one of the participants as it wasn't clear to them what it is and which purpose it has. After

that the question was answered very differently from each person, as the topic was looked upon from the different perspectives of the participants.

The general idea of an SPrL was taken positively but concerns about the practicability were mentioned, because the people weren't sure if such a construct could support the test processes in their respective companies. Most of them stated that a standardized test process which could be applied to every single project of the company would have no benefit as it would have to be very broad and general in its specification. As a second thought they said that it would restrict the development and testing teams too much as it would preset many things for the teams and they had no possibility to change the main aspects. This would mean every team would have to use the same process for every project that was carried out. At Willhaben the current approach is that the teams can decide on their own with which process they want to carry out the project. This enables a more flexible and innovative approach. The participants from the other companies said something similar regarding to the narrowness of a standardized process, as it could potentially limit too much.

As a follow-up question the participants were told to think about the possibility if an SPrL could make sense if it was applied globally. Most of them agreed that it could have made their lives easier when something like it would have been in place when they started to work for their respective company. This was the same answer throughout all persons, so this means a general standardized process for testing could help a newly founded company in getting faster to create a high quality product. Additionally, the overhead of configuring a test process for a company and thinking about all the possible risks are eliminated. The approach of a global SPrL is more likely to be accepted if a company starts, but it gets less likely if the company grows in size. And even if the classifieds companies at Schibsted are working in the same field, they have different implementations and this is hindering if a standardized process that has to be used is introduced. Even if the idea behind an SPrL is that it is adapted, the core process is the same as stated by [83, 65].

Additionally, the SPrL could change the layout of the test process in company. As a new test process is introduced for a project the standardized process helps to get the basic foundation and the experience got from previous projects can be applied to this basis. So an SPrL is helpful when a new company is founded and other companies in the same field of work share their experience to find a common ground. This were the answers of the participants of the interview.

CHAPTER 10

Improving Test Automation at Willhaben

The case study researches the *End-to-End (E2E)* tests on classifieds web pages and checks the current status of E2E tests. This is done by evaluating the state of practice before the implementation of the E2E tests and comparing this to the current situation, where E2E tests are implemented and used instead of manual regression tests. The case study is only done at Willhaben as the code base and test process is known by the author.

10.1 Test Process Analysis

The first step in the test process was the development of Unit tests after the implementation of new features on each platform. This state of practice is still in place now with the introduction of E2E tests. The developers are responsible to write these tests and as the interview participants from Willhaben said, it is always a priority to get coverage with Unit tests as well as integration tests. Those integration tests are also written by the developers, but these tests are often done when a certain part of a platform fails. These platforms are the desktop variant of the homepage, the mobile version, the Android and iOS applications and the jobs platform. When a feature was fully developed the *Quality Assurance (QA)* team took over with testing. This included manual regression testing as well as feature testing, load testing and other kinds of testing.

Manual regression tests were done by the QA of Willhaben, before a test automation team was established. Their was a test plan for every platform. Every critical part was tested before each release, with a special focus on the features that were developed and changes that were made to the different code bases. The regression tests were done with a test plan that had to be executed every time for every platform. This is, as said by the interview participants a very tedious job, as it is repetitive and can include a lot

of possible combinations of *Operating Systems (OS)* and for example browsers for the desktop variant. A small overview of possible browser variants can be seen in figure 10.1[1]. These browsers are just an example to showcase the task for manual software testers to cover as many as possible browser and OS combinations before the release has to be done.

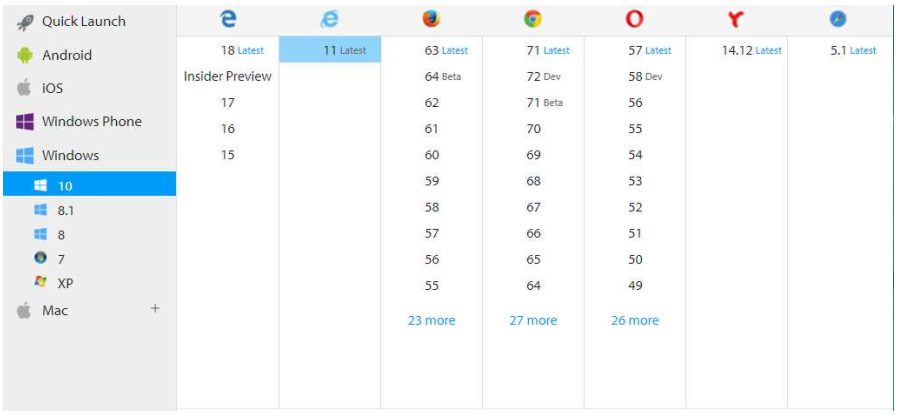

Figure 10.1: Selection of possible web browsers on Windows 10

The QA team at Willhaben additionally had and still has the responsibility for load tests and usability tests. Especially the usability tests are one of the few areas where automated software tests aren't possible as those tests can't evaluate the flow of an application. In [88] the authors describe that the tools they have used didn't provide solutions but can only perform evaluation of the device.

In the end of the test process the QA team had to do the acceptance tests, which consisted of a smaller test set that was tailored down to the platform they were testing on and they also had to do system tests where the API for instance was tested as well.

As stated in section 1.1 the test process included a lot of manual testing, which is now reduced by the introduction of test automation. In the next section the improved test process is explained and how it could be used in other companies as well.

10.2 Improved Test Automation Process

The current test process is shown in figure 10.2. In the beginning the *Objective Key Results (OKR)* and product management input are the driving force behind a feature specification. The OKR's are company specific goals that are set to be achieved by the whole company, depending on how much impact one department has on a specific goal. The feature is specified and from those specifications test hints and specifications for the tests are created. If you look at the test process and compare it with the first graphic

[1] https://live.browserstack.com/dashboard Accessed:16.12.2018

1.1, you will notice that some steps have been added. These include a more detailed separation of the test process into the individual steps and additional steps such as test automation, which has been added.

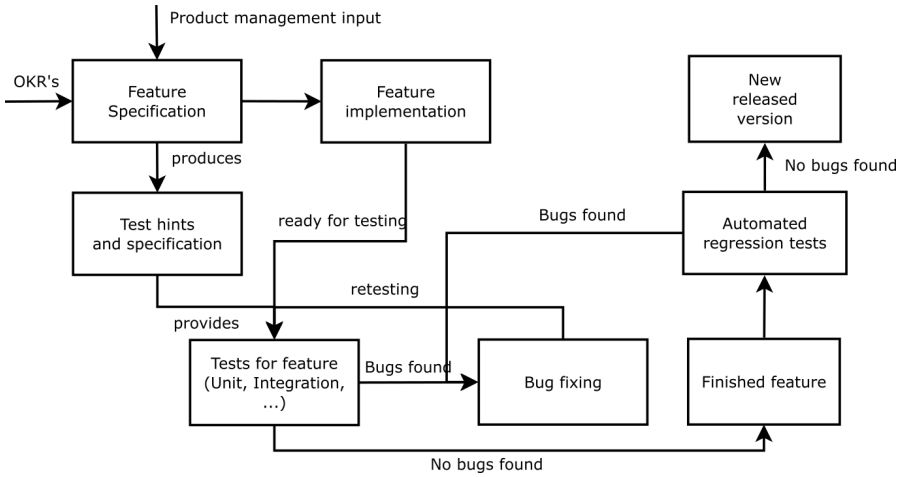

Figure 10.2: Current test process at Willhaben

When the feature is implemented tests are written for it, at a basis the unit tests, which are mandatory for every single feature and integration tests, when a specific part of the system has been newly implemented. There are also manual tests to test if the requirements and specifications of the feature that were written down in the beginning were met. This isn't shown in the process as it doesn't affect the testing because after a new adaptation the feature is always tested. If bugs are found for a single feature a bug report is written and the bug has to be fixed by the developers. After the fix is implemented the feature is retested and if no new bugs were introduced the finished feature is tested with the automated regression tests. Those include a lot of tests as they replace the manual regression tests in the testing cycle. If bugs are found again or something else breaks the cycle starts again with the bug fixing. If no bugs are found the feature can be released.

The E2E tests have replaced the regression tests in most projects, but as said earlier one major project is still missing which are the mobile applications. There are some tests but those aren't enough to check the whole app and therefore are only used for partial checks, when the manual testers can't do everything. This means that even when the E2E tests aren't sufficient for replacing manual regression tests they are still useful to save the testers' time. And this is one effect the E2E tests have on the testing culture at Willhaben. They reduce the load of work for the manual testers and those testers can rely on the test results that are provided. On the other side the test automation engineers have to be aware that the maintenance of the tests doesn't take away the benefits of having them. If the maintenance time is too great, the tests have to be reevaluated and they have to be regularly checked if what they test makes sense. This was mentioned

by some of the participants of the interviews, as everyone that worked with E2E tests experienced that they age as well as the software they are written for. The maintenance of the tests is one aspect that has to be calculated because it can cause problems in the value added to a project. The maintenance for the E2E tests got time intensive at Willhaben and the test team only focused on fixing bugs, which were introduced faster than they were fixed. This is an endless loop which raises the question when there are too many E2E tests, which is also discussed in [18]. One solution for this problem was to move from the development environment to the user acceptance environment. This didn't change as quickly and was more error prone.

The E2E tests are now the tool to check a release branch before it is deployed to production and reduces the amount of testing that has to be done by the manual testers to the open bug tickets. This reduces the time between releases as the testers can focus on edge cases. The time reduction is noticeable and for an ongoing project/product such as Willhaben, the introduction of test automation to a greater degree is, as said by most people from the interviews, a benefit.

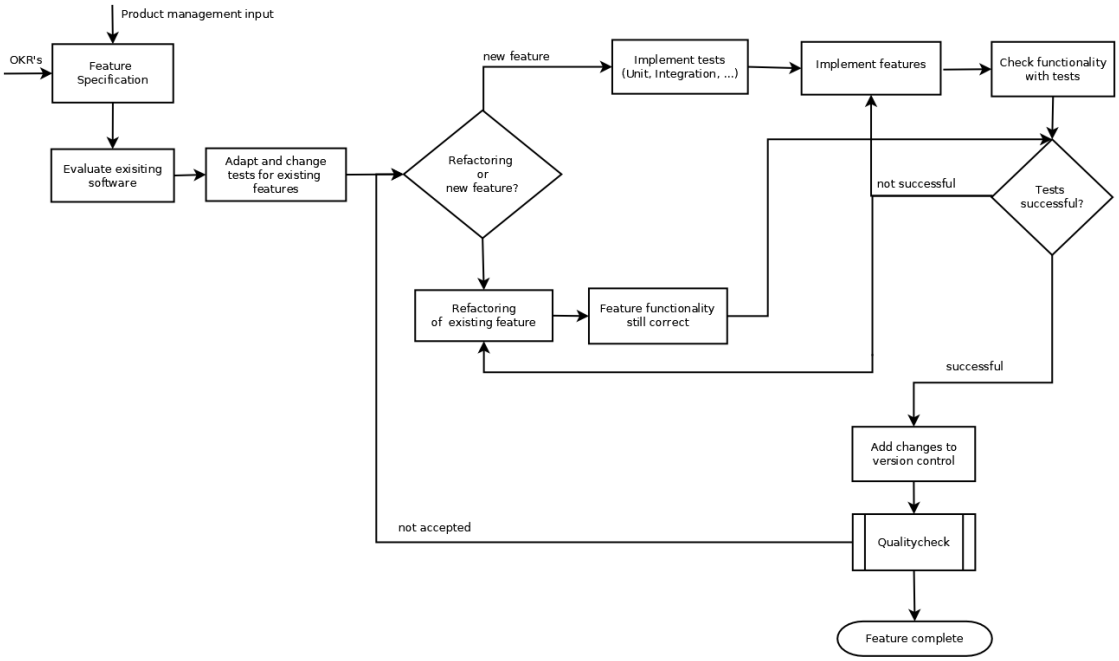

Figure 10.3: Improved test process with the concept of *Test-Driven-Development (TDD)*

This testing process could be further improved by starting the development of the tests, regardless of what kind, before the development of the feature. This would work best with unit tests, and would be the approach with TDD that was introduced earlier in 2.2.5. This development method also affects the test automation and get developers, as well as testers, to write more tests. As can be seen in the survey, many companies are already agile and this supports TDD, but few are really working with the TDD principle.

It can be seen in the different graphs how often the different types of testing are still used and there is still huge improvement potential with the integration and E2E tests. This is shown by figure 8.23 as the majority of companies still use manual tests for verification of a feature. An even further improved test process is shown in figure 10.3. Looking at the survey results, it quickly becomes apparent that an agile testing process like TDD would offer itself to be implemented. In theory, earlier test implementation is often seen as an effective way to reduce development costs across the board [57, 92]. Figure 8.5 shows that most companies use an agile development process and with figure 8.6 it is shown that software testing is already a part of the project planning in most cases. TDD could help here, and for everybody that doesn't include software testing in the project planning phase, it would be good to get field reports about the successes and the most common sources of error as soon as you move.

If you take a closer look at the graphic you can see that the testing process starts earlier and always takes place before the development of the actual feature. In addition, the automatic verification with unit, integration and end-to-end tests happens earlier. The quality check block is a process step that can also be integrated into the previous graphic and is a combination of various quality assurance functions, such as peer review, pair programming or pull requests reviews. This is also done at Willhaben and makes it possible to maintain high code quality. By a high quality standard one can release faster because the process of testing becomes shorter. This is further improved by automated testing, if used correctly.

Another approach is *Behavior-Driven-Development (BDD)*, which has already been described before. This approach would already start with the feature specification and determine the desired behavior there. Even if this approach is desirable, the switch to TDD is already a long way off and would simplify the work of the testers a lot. A unified test process in all projects, and if one considers the requirements globally, also over several companies, would bring a strong improvement here. But one would have to look at this in a follow-up study to be able to investigate the effects of an implemented test process.

The approach presented here must also be questioned with regard to feasibility and cost-benefit analysis. TDD would require developer commitment as the current testing process requires QA to do the testing, except for white-box tests which are covered by unit tests as described in [11]. If you commit to TDD now, this would mean that the developers would also have to take care of black box tests that include E2E tests. Initially changing the process would probably be very costly in a company like Willhaben, as you get a new identity in terms of testing. Changes on such a large scale are easier for companies that are only at the beginning of development to implement. One way to integrate this process into all teams is to first do it in only one team and gain experience. Afterwards, by using a software process line [99, 83], a standardized process tailored to the needs of the company can be used in all other teams.

Even more change would come about through the use of BDD, as you would have to add an additional layer of complexity. The advantages of this would be that *Product Management*

(PM) could also write test drafts. They could generate a test case immediately from a feature specification and then leave it to the developers as part of their work. This change would make PM even more involved in the process and they could quickly encounter difficulties and remove them.

10.3 Evaluation of E2E Tests

In this section the E2E tests at Willhaben, which is a classifieds page, are analysed and investigated. This case study provides some insights on how E2E tests are planned, designed and implemented, as well as improvement approaches to build E2E tests in other companies as well.

At Willhaben the E2E tests, as already stated in section 10.2, are used to replace the manual regression tests. They are developed with Java, JUnit as the test framework and Selenium as the framework for the interactions between the code and the browser. The tickets for the tests are written in JIRA, which are then selected by the test automation team by priority and project. The E2E tests are written for every project at Willhaben and provide a report for each of these projects with the test report after the tests are executed on the build server.

```
public class HomePage extends AbstractPage {
        ...

        @FindBy(id = "wh-aza-btn-big")
        private WebElement azaBigBtn;

        public HomePage(SeleniumProvider seleniumProvider) {
                super(seleniumProvider);
        }

        public HomePage(WebDriver driver){
                super(driver);
        }

        public static HomePage
            open(SeleniumProvider seleniumProvider) {
                seleniumProvider.getWebDriver()
                    .get(getBaseUrl().toString());
                return new HomePage(seleniumProvider);
        }

        public static HomePage open(WebDriver driver) {
                driver.get(getBaseUrl().toString());
                return new HomePage(driver);
        }

        public void initPage() {
                require(willhabenCodeTextField).visible();
        }

        public String getTitle() {
                require(azaBigBtn).clickable();
                return getWebDriver().getTitle();
        }

        ...
}
```

Listing 10.1: Homepage of Willhaben with the PageObject Pattern

The E2E tests are developed in the page object pattern, which was described already in 2.3.3. The pages are used to separate the test case from the logic behind the page. An example page is shown in the previous code example, which is from the code repository from Willhaben.

The page object pattern can be a general rule for different teams, platforms and systems as it is only a design pattern, which can be applied on any web page or even mobile application. This enables the teams to work together even if they don't know the tests, as they should be interchangeable. This can even be used if the teams aren't in the same office as the principal is always the same. This design measure helps when different people have to go through the tests and make changes. Due to a uniform structure, the test procedure is always the same and this can be exploited in a global collaboration.

The JUnit framework is used instead of TestNG which was used before as well, but was removed out of simplification. The usage of two different but also rather similar test frameworks doesn't provide benefits, as stated by [120]. Therefore only one framework is better as it reduces complexity, time constraints when switching between the two and needed knowledge. As each of the frameworks has its twerks as can be seen in [3, 6], but it is easier to manage only one. Using the same software framework for all tests has been a good decision, because you can use the existing expertise in the company to write better tests. This also helps when implementing this measure in a project that involves more than just the local *Product Development (PD)* team. Even if the tests are written in different languages, but behind them the same framework is used for verification, this helps the developers.

The reduction of manual regression tests and efforts put into repetitive and tedious tasks is one of the main goals of E2E tests at Willhaben, which was mentioned in the interviews. Another big part was increasing the speed in which a release could be tested. The reduction of time spent verifying already implemented features with E2E tests is also in the focus. This was also mentioned from other interview partners, as their goal with E2E tests is to keep manual testing to the bare minimum. This was achieved by Subito, which is the Italian pendant to Willhaben. There the E2E tests are more evolved as they have tests for the mobile applications, which isn't the case at Willhaben. The interviewees stated that the tests are sufficient enough for them to rely completely on them. Another important aspect there is that they have a great unit test coverage, as well as enough integration tests that the E2E tests complement them. In other companies the current state is that the implementation of E2E tests has only started but the overall goal is the same.

For the manual testers and developers it is important to get reliable test results from the E2E tests, as it doesn't help if flaky tests mask real errors or everyone is used to one or two failing tests in each test run. This is a problem with E2E tests as they rely on many components when they are executed on a complex system, which is the case at Willhaben.

Through the use of E2E tests, the proportion of manual tests in the company should

decrease and no longer be the dominant type of testing as shown in figure 8.13. However, it is important to emphasize that this is not a complete abandonment of manual testing, but only a reduction of this type of testing. TDD would also bring unit testing more to the fore. As mentioned in [12] it is critical to select the correct tests for E2E tests.

CHAPTER 11

Discussion

In this chapter the results of this work are discussed and compared to other results found by researches from the state of the art chapter. Those results are aligned with the research questions that have been asked. The collected data from the use case, survey and the interviews is discussed and summarized. The case study about the current test process and the automated *End-to-End (E2E)* tests is used as well to get an overview on what could be improved even further with an adapted test process. The catalogue of improvements that summarized some of the possible improvements of the test process at Willhaben has to be discussed as well. After that the limitations for this work are discussed.

11.1 Best practices (RQ.1)

In the first *Research Question (RQ)* the focus is on best practices for software testing. Here the survey results on different practices that are done in various companies are compared to actual best practices according to literature. The catalogue of improvements is used additionally to find areas where a best practice would help to improve testing.

One best practice for software development in recent years was to switch from traditional software development teams to agile ones. This is underlined by the survey results on how many people are still in traditional teams. Therefore, the numbers of people in agile teams in comparison to traditional ones can be explained by the trend to switch to a more agile approach in software development as stated by [36, 42]. The traditional approach has some downsides in comparison to agile development practices, as it is slower to react to changes in the specification and testing is done later in a traditional team [35, 90]. The main agile method that is used in most of the teams is SCRUM, or as many stated it's close to SCRUM but an adaptation to the needs of the company.

Software testing should be, as stated by [57, 43], part of the software development cycle if the team is working in an agile environment such as SCRUM, extreme programming or Kanban. As soon as a project is planned and the first feature specifications are done, there should be input from testers and developers, to improve the quality of these tickets. [43, 36] Additionally, both can tell the product manager if something is even doable or testable. This was also mentioned by an interview participant, because it isn't the state of art at Willhaben for every project. The earlier a tester is involved in the project planning phase, the easier it is to change requirements in such a way they are feasible to be tested. In the survey 77,4% said that they have software testing in the project planning cycle which underlines the importance of software testing. Therefore it should be a best practice to include software testers and developers in the project planning and feature specification process.

The inclusion of software tests in the development process is the next best practice that has to be included when considering software testing in a company. Especially unit and integration tests are included by default in most agile processes [36, 42, 43], but E2E tests aren't as often used in projects as can be seen from the survey. Here only half of the survey participants stated that E2E tests are used frequently or even in every project. The importance of E2E tests especially for a company like Willhaben, which has a lot of user interfaces on a web page, is increasing. Testing these web interfaces as a software tester with manual regression tests is tedious and repetitive. This can be achieved faster and more thoroughly with E2E tests [17]. The focus on more automated tests, which provides a lot of advantages [12], as described by [17, 50, 30] hasn't reached its full potential at Willhaben and the other companies in the Schibsted. This is shown by the high amount of people saying that manual tests is still the main focus of software testing. This is especially surprising as it is only beaten by a small margin by unit testing, which should be the clear number one, as every feature should have unit tests that cover this feature.

Test-Driven-Development (TDD) or *Behaviour-Driven-Development (BDD)* are approaches to improve software testing even further as described in [57, 31, 30]. These approaches could be considered best practices for software testing and *Quality Assurance (QA)* as well. In [57] the authors describe that TDD and BDD are enabling QA to be brought to a higher standard. In figure 8.24 the scope of manual testing is described and here the majority, if answers are added up, says that the tests try to detect defects and bugs rather than preventing them, which is called quality checking. The prevention of bugs would be quality assurance, which is the case when many unit, integration and E2E tests are done. Those tests provide early detection of bugs and those bugs never get close to production, as they are caught way before. When TDD or BDD is done the prevention rate is even higher compared to traditional software testing. In interviews some of the participants where asked if it would make sense to switch to one of those methods. Most of them stated that the methods would strongly interfere with the already established work flow, but that it could be applied to new teams. Here the individual responsibility of each person in an agile team comes into place, as they would have to want to use TDD

or BDD for their team.

The use of similar programming languages or frameworks for the various tests is a suggestion for improvement that we have determined through the practical test. If you use multiple frameworks to test the same functionality, it makes setup unnecessarily complicated and prevents you from efficiently reusing existing structures [120]. This leads to a careless and complex handling of the frameworks, which makes it easier to incorporate errors. This was one reason why Willhaben decided to change from TestNG to JUnit and removing Cucumber with Calabash and introducing Appium instead. As stated in [100] the usage of different languages should be avoided, if it's not absolutely necessary. On a global scale this is not enforceable and one should not break the grown architecture only to use the same programming language everywhere. With the frameworks you could agree to use the language specific ones, but they don't differ too much in structure. Design approaches like for example the page object pattern can be set as standard for everyone instead [45, 46].

In [21], the annual cost of fixing a test in the different phases of development is shown and the curve increases exponentially, as shown in figure 1.3. Therefore, in software development you should always make sure that testing is not neglected. In this context, we would like to refer again to figure 1.2, which represents the ideal testing pyramid.

Looking at the survey results from table 8.2, it becomes clear that manual testing has a very strong influence on companies, but should only be used as an additional factor in quality assurance. Unit tests should always be used, but sometimes they are rarely used [64]. Integration tests are the second big issue and, according to the survey, they are far too rarely used. This stands in strong contrast to the ideal way, as the use is strongly recommended [52, 48]. E2E tests should only be used for critical functions in the program [52] and this is in line with the survey results. You could increase the number, but it's good that E2E tests are used at all. The best practice here is that the usage of Unit tests should be high and if we go through the pyramid, the usage of the other test groups can be less. Manual testing can't be completely eliminated and shouldn't be, but the reduction of manual testers in comparison to automated software testing is the way to go.

11.2 Requirements for E2E tests (RQ.2)

In this section the second RQ is discussed and how requirements for E2E tests from a company relate to those brought up in theory. The efficiency and effective usage of E2E are main focus points.

One requirement for an effective usage of E2E tests is that the developers and testers, that are affected by E2E tests, are familiar with them. This is the case at Willhaben and Schibsted, as the percentage of people that know that there are E2E tests used in their respective company is high as can be seen in figure 8.25. A lot of people stated that they are actively participating in the test process which can be seen in figure 8.14. This helps

as well, as they already have expertise in other testing techniques, even if they don't develop E2E tests. In figure 8.28 the responsibilities of who writes E2E tests are shown. QA is the mainspring but developers are also a key factor for success. This is shown in [49] as developers need to know how to test and how to write test cases. Developers shouldn't be the only source for testing, especially if they test their own code, but they can provide another viewpoint in comparison to the testers [49].

As can be seen in table 8.16 and 8.3 there is a wide variety of programming languages used in the different companies, which can be explained with the different platforms that are used throughout the Schibsted classifieds. The different technology stacks offer a wide range, but this causes a problem as well, because it demands separate testing teams in every single company. Additionally, the software can't be combined easily and a single codebase for all the companies doesn't work as well. When the same E2E tests should be used throughout Schibsted the different platforms would need the same language or at least the same layout in the different web pages. This isn't the case, but as more and more micro services for every company are rolled out it would be possible to at least bring the platforms closer together on those components. A global test automation isn't possible, but when the platforms move closer together regarding technology stack and user interfaces, it could be possible to use already written tests in another company as well. In [28] the author worked for a company that had very few test cases, which was also the case at Willhaben and is still the case in many companies in Schibsted, that are essentially just beginning to develop the classifieds web page. Those newer projects could be set up so that the structure and the knowledge of how to do E2E tests is shared and synergies are used between the different companies.

Another requirement for successful development and use of E2E tests is the coverage of the tests and if those tests are sufficient enough for the development teams as well as QA. A pre-requirement for this is that the most important business cases and critical paths through the *System Under Test (SUT)* are analysed and written down. This is done at Willhaben as stated by the interview partners and by analysing the test plans. In [28] a similar approach is taken, as the manual tests that were done before are written down and used for the initial set of automated tests. Those tests are constantly extended which increases the coverage and therefore the usefulness of the E2E tests. In figure 8.18 the satisfaction with the amount of testing in general is shown. As said before in section 8.2 most of the survey participants are satisfied with the amount of tests which are being executed, which doesn't mean that E2E tests are included as well, but it can be assumed that the participants from companies that use E2E tests as well are satisfied with those tests as well. This is shown in figure 8.25 where 70% said that they use E2E tests, which correlates to the value of figure 8.18.

In [28] the author states that there are two ways of testing the UI which he knows. The one way is to have starting points from where only one specific feature is tested. The other way is to have a main path which every test more or less uses and the single tests branch into different paths like in a tree design. As described in [28] there can be several problems with the later approach, as many unrelated failures to the specific test case can

occur on the main path. If E2E tests are only started where the newly developed feature starts, those unrelated failures can't occur, but on the other hand the test doesn't test the surroundings features. At Willhaben a sort of the tree design is used, which causes problems in the test development, as many failures can occur, that have nothing to do with the original feature that should be tested. Therefore, another approach was taken for some of the tests, to reduce the amount of failures. This was done with well designed tests that are short and don't cover too many features at once. This way the error rate in those tests is way less than in the other ones and therefore the maintenance is lower as well. This should be considered when a new project with E2E tests is set up and started.

In the previous section, the use of a uniform design pattern was suggested as best practice. This also applies to E2E tests and must of course be seen as a requirement for these tests. An example of a uniform design pattern would be the Page Object Pattern, which is often used in E2E tests and has already been described in chapter 2. This approach describes individual pages as objects and can be easily transferred to other web pages by this procedure, because every page of a homepage can be mapped to such a page object. If you use this design pattern it is also easy to understand for people who don't have to deal with the tests on a daily basis [45, 46]. When we analysed the page objects at Willhaben and asked the people how understandable the mapped pages were in comparison to tests, which just put the whole code in the test file, the answers were that they were more readable and usable if a test had failed and the reason behind it had to be checked. This finding coincides with [101] which explored the influence of the page object pattern on maintainability and readability.

11.3 Catalogue of Improvements

In this section we construct a catalogue of improvements for the test process at Willhaben as well as requirements for successful development of E2E tests on the results of the case study, the expert interviews and the survey which were conducted previously.

One of the actions to improve the performance of the test process at Willhaben is to define better specification tickets to give the manual testers, who do feature tests and verification of those specified features, a higher possibility to be faster with those tests. This was stated by one of the interview partners. A well written feature specification additionally helps the developer to provide better quality work products that have to be checked.

Another approach to improve the test process is to include the testers and developers in the feature selection and design phase, to get early feedback from them. The current process at Willhaben doesn't support such a behaviour, as the specification is solely written by the product manager. Afterwards the developers get the feature request, which should be finished, but this isn't always the case, as the feature isn't fully specified. This causes delays and implementations that are based on assumptions. When the tester is than confronted with this ticket, he/she can't rely on the feature description alone, as the implementation could be different. This can especially happen when the

same functionality is implemented differently on the different platforms. This could be mitigated when a base approach is taken to get a fully specified feature request and if the tester is included when the feature is specified.

Writing more test automation tickets to showcase the need for an additional test in a specific test scenario to the test automation team is another improvement point. The testers write tickets if they find a new bug, but this could also be done by the developers, to further improve the test coverage as well as the reliability of the tests. The tickets for E2E tests can be done from feature tickets, as those tests do test a feature within a certain path. The feature request is a good starting point from where a new test can be created.

To improve the coverage overall for all different parts of the software, the test process should be adapted to have better checks of verification for unit and integration tests. As the use case has shown, there are unit tests and integration tests written, but especially integration tests don't have the coverage they should have as shown in figure 1.2. Here better metrics and monitoring could increase the will to write more tests in general. Load and smoke tests have to be written as well.

When the results of the survey are considered, some of them said that there can't be enough test automation and some said that their current process doesn't include enough test automation. This also includes unit and integration tests which have to prioritised by management to improve overall reliability of tests for the whole company.

Another approach is to switch to TDD [57]. This is solely for unit tests but it helps to get faster responses on how a feature performs and how well its specification is implemented. In [92] the author lists some of the advantages of TDD, like high test coverage, which is important to rely on the tests. Another factor is the maintenance and refactoring of code which is well tested. But there are also disadvantages, such as every one of the team has to be committed to do testing. This was critiqued by one interview partner as it takes away the freedom of the developer and can be hindering. Others stated that TDD could be used, but it would require commitment from everybody, especially from product management, as tests take away feature time, but this would be outweighed by the benefits if we look at [57, 92].

The test process should also re-evaluate legacy code which isn't done at the moment at Willhaben. Those legacy code projects are the ones that cause many problems as they are often not well documented or hard to understand. Those projects are often not well tested and a change can cause major issues in other parts where it isn't expected. Additionally, the responsibilities of project owner ship in the test process should be more specific for projects as this leads to moving around responsibilities. When this happens the project isn't thoroughly tested and here a clear documentation on who owns a certain project and is responsible for it, would be an improvement.

11.4 Improved Test Process (RQ.3)

Here the test process improvements for the analysed *Software Process Improvement and Capability (SPICE)* processes are discussed and checked how valid the approach is. The use case from Willhaben provides data to work with as well as the expert interviews and survey results on the test process in the different companies.

11.4.1 SPICE capability levels

When the design science method is checked, the goal is to improve a specified RQ and for RQ 3 the question was how to improve the test process at Willhaben, where the use case was specified. The test process regarding the different process groups in SPICE are on different levels and can therefor be improved to achieve a higher level. As not every process group is touched in this work an overall SPICE level for the organisation can't be delivered, but the most important process groups for the test process are rated. [82] The selection of the process groups was made after each has been evaluated for meaning and content.

The first process group is software unit verification and it is classified as being on capability level 3, while level 4 wasn't fully achieved as described in 7.2.1. To reach level 4, testing must be built into the company's objectives and must always be set as a goal. If you want to achieve this at Willhaben, it is necessary to include the time for tests in the planning and not only for unit tests but also for all other test types described in chapter 2. In addition, the amount of information needed about the process to support the business objectives would need to be determined as written in [82]. This would result in the objectives of the process measurement and this would mean for Willhaben that after the processes are evaluated the expected results would occur. This is partly already done, because company goals are measured and the stakeholders are also asked about their needs, because these of course set the company goals, but in relation to tests this does not happen enough. Testers are only partially involved and tend to play a subordinate role. If the relationship between the individual test categories and the software being developed can then be established, with stakeholders such as the testers on board, SWE.4 could be raised to capability level 4 in the quantitative analysis process attribute. In addition, monitoring, analysis and verification measures would have to be taken. For this you can use existing systems that also use software development.

For achieving level 4 completely the quantitative control process attribute has to be fully achieved as well. In order to analyse the collected data, the existing structures would have to be expanded. There are simple reports that can be sent to everyone, but problems with tests are usually only taken seriously if a real bug has occurred. Failed tests that have happened due to a feature change are only improved when there is time. This would have to happen immediately, otherwise deviations from the process are not noticed. These changes would raise the process to capability level 4 as described in [82].

The second process group was software qualification test, which is on capability level 4, which is the predictable process. Therefore, the next and final stage would be level 5, the

innovating process. The process is already at capability level 5 in some areas, but not in all of them. Positive examples are, for example, the definition of process innovation goals for the business goals, as provision is already made for the future by using test automation to improve regression testing and reduce the effort spent on it. For this purpose, the data is also analysed, because for example the time a tester needs to test everything is recorded. In addition, the resource "tester" is also analyzed and checked to see if it could be better used elsewhere. This is also done in [12] and there you can see how the time expenditure changes and also the tasks of a tester if you can use him/her for other areas.

Areas where Willhaben has to improve to reach capability level 5 are definitely the analysis of new technologies, also for manual regression testing, because unfortunately this is not pushed in normal operations. The evaluation of a proposed change to the standardised process would also need to be done, as this is rarely done so far.

As the third process group quality assurance was analysed and checked on which capability level it is according to [82]. Quality assurance is at level 3, and therefore would need the same changes as the first researched process group. As written in section 7.3, the largest problem with achieving level 4 is that tools for analysing the process data aren't used. In addition, no measures are taken to address the causes of deviations. Periodical checks have to be implemented as well, to get results from the process faster and to react more quickly to necessary changes.

Verification was the next and it is on level 3 as well with the potential to be on level 4 if some improvements are done. As described before in section 7.3, a change necessary to reach capability level 4 is that all teams align their verification processes with the business objectives. The different teams are at different levels in this context, and if you weren't looking at Willhaben but only at the App Team, Willhaben would be further with this process.

As the last process group to categorize for the software process the process group process improvement was checked. This is the lowest of all analysed groups as it is only on level 1 and therefore has the highest potential to be improved. This would also result in the strategy for implementing innovations. If you want to establish a managed test process, which is constantly improved, at Willhaben, you have to define the targets for the performance. This happens in some areas, such as test automation, but there is no standard for how a test should look like or what requirements are given. In addition, the test process itself is not monitored, but only periodically checked to see whether it still makes sense. If you want to establish a managed test process, which is constantly improved, at Willhaben, you have to define the targets for the performance. This happens in some areas, such as test automation, but there is no standard for how a test should look or what requirements are given. In addition, the test process itself is not monitored, but only periodically checked to see whether it still makes sense. The documentation of the set steps could simply be taken over by other processes, in addition one could also use the improved test process shown in this work to measure the progress. Improvements to the originally conceived process could be written down. Many of the steps you would

need to get the process improvement to level 2 or higher are already being done in many other process groups and just need to be copied.

11.4.2 Using a Software Process Line

The *Software Process Line (SPrL)* was discussed as a second focus point for RQ 3 with the use case at Willhaben. The interviews provide some data on this topic as one of the main questions was about SPrL's. This question was about the usefulness of SPrL's in global teams, which had to be explained to all interview partners, as they had never heard of this method before.

The general idea was taken positively by the interview partners, but the implementation of such an SPrL is thought to be as problematic, as the process line would be very general. This problem is described in [65] as the SPrL has to implement tailoring rules to comprehend with the variance of the different processes. In [65] a megamodel is presented to build an SPrL and the authors used it for several projects, but the initial setup costs are a major factor. As every company has their defined set of processes, which were tailored to the specific needs of the company, it would need further research if an SPrL would make sense and wouldn't cost too much to setup.

If we were to apply testing to all teams in the same way and use a uniform process, a lot of overhead would be eliminated on the one hand. On the other hand, the freedom to decide whether to use a certain process would no longer lie with the teams but with the process line. This was noted several times in the interviews, but this freedom should not be restricted. For the testers it would be a lot easier, especially if, for example, the developers were also responsible for testing. A possible approach to this would be the whole team approach, which puts exactly this into the responsibility of the team. A general process line would help to define and standardize the guidelines by which each team is measured. This was said in the interviews especially by the testers. This standardization is also described by [65] as well as by [99].

The positives of the implementation of such an SPrL on a global level for the testing process are thought to be there, but only if it is done right in the beginning of a project or the founding of the respective company. At the start of a new project it might be useful, but as said earlier the overhead of setting up such a huge project, where the outcome is uncertain isn't feasible for a company such as Willhaben and Schibsted on a global scale. The approach could be used for different things and one approach could be that the SPrL isn't just used for the software testing process, but for the development process as well. But right now the time for that, at least for Willhaben, isn't here yet.

CHAPTER 12

Limitations & Threats to Validity

In this section the limitations of this work are explained and how those limitations effected the work on this subject. There are some major constraints regarding this work, as not everything, that was mentioned in previous chapters can be covered. This has to be done in future researches, as it would take too much time to cover all details. For this purpose limitations to the validity are written down, so that this work can be used in a broader spectrum. As stated by [60], there are four main threats to validity. Those are conclusion, internal, construct and external. The threats to validity will be mainly based on [60]

12.1 External Threats

In this work the best practices for test automation are researched. This could be done in nearly every field that has something to do with software testing, as every company that develops any kind of software should have testing in place. Software testing should always include test automation, even if it just includes the basic Unit and Integration testing. Therefore, the first limitation for this work is that the field of research only refers to companies that are classifieds ads platforms. Those platforms provide marketplaces to buy and sell stuff for both professionals and casual users. These companies can be located all over the world as this isn't restricted, but they have to be in Schibsted. This is important because Schibsted allows access to the needed information and grants the researcher the right to publish the outcome of this research. This is supported by the fact that the researcher works for one of the companies of Schibsted, as explained in chapter 1. Additionally marketplaces have a huge impact on many people and testing helps the users to have a better experience and usability of the websites. Adding test automation to the quality assurance is a step forward in improving the sites even further. This concept can be copied and used for many more applications but evaluating more than the previous specified companies would take too much time and would increase the

complexity and length of this thesis. Therefore, only companies that belong to Schibsted and are classifieds ads platforms are considered and are evaluated in the survey that is proposed in the next chapter.

The next limitation is that only certain frameworks are considered in the survey and in the work as well. This includes Selenium, Appium, Calabash and others mentioned in 2.3. There are many others that could be better suited for a specific project or problem. This list of frameworks where those that are most popular or used in the researches' company. As the survey shows many people use different frameworks or tools to apply test automation to there projects. The de-facto standard for end-to-end tests as stated by [39] is Selenium. Its wide use and importance for other frameworks or tools are the factors why it's selected for this work as well.

Additionally, this research is focused on web/mobile applications and not on desktop applications. This could mean that the results are not 100 percent d'accord if applied to other areas. It can be that minor changes to the recommendations have to be made to be able to achieve the same result.

Another limitation that has to be taken into account are the countries that were considered. The companies are located all around the world, but the results were compared to an Austrian company, where work labour is expensive and working hours are regulated very strictly as can be read in [66]. This means that if a similar research is conducted the local law and commonalities have to be deliberated.

Another point to consider here is that the group of participants are all working in the same field of work, which doesn't have to be the case for the survey, as it could be given to a more diverse group as well. If a lot of different companies with different fields of work are asked the results will probably differ in some way or the other.

The survey itself has an effective size of 63 participants and this is quite a small number of people. Furthermore, the group of people that were asked to do the survey was especially monogamous, as it primarily consisted of people that are working in tech or have a strong affinity for technology, such as the product management. The results therefor can be different if another group is asked to do the survey, as it's tailored down to fit a tech-savvy group. If a larger and more diverse group is considered, the questions have to be rephrased or completely changed.

12.2 Internal Threats

A limitation is that the results of this research aren't necessarily used in the companies that the research was done in. The results include a catalogue of improvements that can be used to improve the overall quality of the software product, but if the recent quality is good enough for the companies, there is no possibility to enforce the catalogue. It is just a recommendation and therefore no further research on how the improvements are implemented in every company is done. The list of recommendations are optional and can be used if the company decides to do so.

Another one is that the timing of the survey and the implementation of best practices can be far apart. This means that requirements and factors may change and the benefits may no longer be the same as at the time of the interview/survey.

The test experts in the interviews were selected specifically for different reasons, such as experience, answers on the survey and availability, and not randomly, which is a limitation to the gathered results from the interviews. The opinion of the person on certain topics always changes the answers that they give and the culture in the companies influences that as well. The results from the interviews will be different if other people are selected. This does also include that the outcome of the interview could be different if the same person is questioned after a certain period of time. In this period something could change in the company and therefore the answers would be different.

The interviews were also done so that the answers of the survey influenced the questions that were asked. Additionally, the participants of the interviews weren't all part of the survey, but those people were also asked some of the questions that were asked in the survey, to get some basis. The interview questions therefore differ quite a lot, as the interviewer had to react to the answers the participants gave.

The case study on end to end tests was only conducted at Willhaben, as time constraints and availability of access to the code bases of the different companies posed quite some challenges. This means that if the case study is used as an example, this has to be considered, as the sample size of the case study is only based on one company.

CHAPTER 13

Conclusion

In this work the test process in a global company was discussed and if it could be improved with an *Software Process Line (SPrL)*. For this a quantitative survey was sent out to different companies that are running a marketplace and as said before those companies belong to Schibsted. Additionally, structured interviews were done with several people, that agreed to answer more questions after the survey. With those two methods and the state of the art best practices were found that could be applied to companies that are working in a similar field, or with some adaptions in other fields as well. Those best practices were collected to use for the test process and an improvement approach with SPICE was discussed and analysed. As a first step the state of the art of software testing and software test process was introduced and elucidated. The main focus was how a test process could be improved and which improvement approaches could be used for this purpose. For the best practices the current state of the art for testing and the test methods were explained and the optimal test pyramid was shown. The SPrL was elucidated as well and the benefits of such a line were presented.

13.1 Lessons Learned and Cost Benefit Analysis

As written in the intro of this work, chapter 1, one goal is to elaborate best practices for testing with *End-to-End (E2E)* tests, but also with other test types. This includes unit, integration and regression tests. Best practices involve, as stated in section 11.1, the switch from waterfall to more agile approaches on software development [95]. Even if a company doesn't use textbook SCRUM, they can react more quickly to changes. Another best practice found is the involvement of software tester in the project planning phase, instead of presenting them a complete feature. With this change, they can bring in their expertise right from the start and help identify complex issues and possible weaknesses or sources of error.

The approach taken was to use the design science methodology to answer the research questions that were raised through the problem statement and the state of the art. The state of the art couldn't solve every problem and therefore other methods had to be used, such as the survey, the structured interview and the analysis of those results with SPICE. The survey needed enough answers to be usable and the target value was to have 50 to 100 people answering the survey. This was achieved but it was more complicated than initially thought. For the interviews the goal was to have 10 people to answer the questions. It was difficult to get enough people for the survey and the interviews because many didn't feel addressed by the emails that were sent. Even if you point out several times how important many different opinions are, few people are willing to answer something.

When we check the results of the interviews a lot of the answers to the questions were as expected, such as the different programming languages, because a lot of companies were expected to participate. Others on the other hand were surprising, such as the amount of manual testing, which is still used in the different companies. Here we expected a lot less, but manual testing is still a feasible option for companies. The test process at Willhaben or at other marketplaces of the Schibsted and how it sometimes does not exist was an outcome we would not have expected. If one reads through the literature, it is suggested that every company should have at least one. An unstructured test process, which is constantly changing, is not expected in an established company. But this was exactly the result of the first analysis at Willhaben, which changed in the course of the work as more and more points were incorporated.

If one looks at the costs of introducing *End-to-End (E2E)* tests compared to the benefits, it quickly becomes clear that although the costs of introducing E2E are higher, as stated by [21], they quickly become poor. This happens all the faster if you can run the tests often, as Willhaben does by using build servers. *Continuous Integration (CI)* is the key here because you can schedule the tests to run. It doesn't have to be a complete CI system, but the next step would be to run all tests with every build and only release it if they are green [23]. Using a single framework for E2E testing makes sense when looking at the survey results. The most commonly used frameworks are mostly the same as those described in the state of the art [11, 6]. Even if you have different languages as a basis, you can still use the same or very similar frameworks [5]. This is shown in figure 8.29 and in 8.17. The exception here are iOS developers, as they have the support for XCTest. The use of many different languages for writing tests is the freedom that most frameworks give developers. Many of the frameworks are also available in another language, as shown in figure 8.26 for E2E tests and in 8.15 for the project languages, which includes Unit and Integration tests.

The improvement of the testing process is a central point of this work. An analysis of the currently used process of Willhaben was done by the SPICE assessment. The process is at a very high level in most of the process areas examined, except for the process improvement process. This also carries the highest risk, but also the highest potential for improvement. It is easy to overlook the fact that a process, once it has been

implemented, needs to be continuously improved. As seen in the use case, the monitoring of this improvement process is usually neglected.

The implementation of a new test process, which is not based on a grown process, but is planned, is, as already written in section 10.2, tedious and also involves costs. However, if the company is committed to changing one team after another, the new testing process can also be implemented in a larger company. The cost of implementing such a testing process is replaced by reducing the cost of fixing bugs. The initial cost factor is higher, especially for *Test-Driven-Development (TDD)*, because the developers are bound to the tests. However, the longer TDD is used, the faster tests are fixed. Also the development of features becomes faster, because, as described in [57], the features show fewer bugs, which have to be fixed afterwards. As described before in 10.2 *Behaviour-Driven-Development (BDD)* would be another approach that can be used.

The introduction of a software process line for software testing would cause some costs because existing structures would have to be rebuilt and processes that would have to be used would have to be adapted. These costs are, if you don't have massive problems with the existing processes, not worth it to create a new process line. However, if you have a young company or are building new teams, it makes sense to force a standardized process. This is mainly the result of the interviews, as many of the interviewees said, that a new company could be faster if a set of standardized processes are handed to them. This is also underlined by [99], as it checks various companies with different sizes and the difficulties are the same if the company grows in size.

Regarding the requirements for E2E tests, it can be said that the people involved in the tests should also be familiar with them. This applies only to developers and testers in the first step, but if one uses BDD, the understanding of the PM team may also be required. Expertise from manual testers, as described in section 11.2, is of great importance and can help to improve the tests again. This is described in [49] as well.

13.2 Future Work

An additional research could further explore the implementation of the suggestions for improvement and the catalogue that was written down. The research could check if and how the best practices are achieved and if the improvements bring the expected results. Additionally, another research could tackle this issue in other fields of work and if the results from this research can be used for others as well. As mentioned before another work could investigate the effects of a unified test process that is introduced to a network of companies and how this effects the work of the employees and the overall effectiveness of testing as a whole.

In this work an SPrL for software testing was introduced, but the resources and the need for an SPrL weren't given, so in a future work an introduction and a better analysis about how an SPrL could be included in the whole test process and how it could help with generalizing the test process could be done. This would improve the knowledge and

practicability of SPrL in a general matter. As seen in this work, a very general SPrL for the automated tests could help in a global team, that is starting to automate different components and software parts. A different approach was taken by [99], but this work could be the basis for a future work, which includes SPICE instead of CMMI.

One part of E2E tests which wasn't touched at all in this work was visual testing. This is a subset of GUI testing and could be focused on as well in the future. The impact of visual testing on the test process and if it should be separated from the traditional E2E testing could be another study. An empirical study was done on the subject of maintaining those test in [93]. In [94] an approach is described to use behaviour driven development with E2E testing and how it could help to make test automation available for non-developers. This could be done for visual testing as well.

List of Figures

1.1 Release Process at Willhaben 2
1.2 Ideal testing pyramid 4
1.3 Costs of fixing an error from [21] 5

2.1 Caption for LOF 9
2.2 Resource triangle from [20] 13
2.3 Test automation pyramid from [48] 15
2.4 Classification of software testing into levels, methods and types from [11] 16
2.5 Test driven development 24
2.6 Selenium IDE 32
2.7 Appium architecture from [23] 36

3.1 PDCA Cycle from [78] 44
3.2 Automotive SPICE process reference model from [82] 51
3.3 Relationship between assessment indicators and process capability from [82] 54
3.4 Three processes view from [70] 56
3.5 Domain and application engineering from [70] 57

5.1 Challenges within this work 64

6.1 Adapted design science framework from [91] 68
6.2 "Goal structure of a design science research project" R. J. Wieringa, [91] . 68

7.1 Selected process areas from SPICE 74
7.2 Identified vulnerabilities of each process 85

8.1 Survey design with the 6 different blocks 88
8.2 Workplaces of the participants 90
8.3 Experience in the current position and the overall experience in the field of work 91
8.4 The software development process that is used in the different teams 91
8.5 The most used software development process of the participant 92
8.6 Software testing part of the project planning cycle 93
8.7 Unit testing as part of the development life cycle 94

8.8 Integration testing as part of the development life cycle 94
8.9 E2E testing as part of the development life cycle 95
8.10 Regression testing as part of the development life cycle 95
8.11 Manual testing as part of the development life cycle 96
8.12 The estimated testing effort in comparison to the overall development effort 96
8.13 The main type of tests that are used 97
8.14 Familiarity of participants with software testing 97
8.15 Languages used in projects 98
8.16 Languages used in projects 98
8.17 Different frameworks used for developing unit and integration tests 99
8.18 The satisfaction of the participants with the amount of testing that is done 100
8.19 Company employs explicitly manual testers 101
8.20 Manual testers are part of development teams 102
8.21 Manual testing required in various projects 102
8.22 Should be the amount of manual testing be increased? 103
8.23 How often are the manual tests executed? 104
8.24 The scope of manual testing 104
8.25 Usage of E2E or functional tests in various companies 105
8.26 Languages used for developing E2E tests 106
8.27 Table for Q26: Which language are you using for these tests? 106
8.28 Who has the responsibility to write the E2E-tests? 107
8.29 Frameworks used for developing E2E tests 108
8.30 Table for Q28 108

10.1 Selection of possible web browsers on Windows 10 116
10.2 Current test process at Willhaben 117
10.3 Improved test process with the concept of *Test-Driven-Development (TDD)* 118

List of Tables

2.1 Annotations in TestNG. This table is from [6] 29

3.1 Comparison of Capability and Maturity Levels from [73] 47
3.2 The different capability levels of ISO 15504 from [82] 50

7.1 SPICE capability levels of the researched processes 85

8.1 Table of Q8 summarized 92
8.2 Table for question 10 of the survey 93
8.3 Other programming languages that are used 99
8.4 Frameworks used for developing unit and integration tests 100
8.5 Percentages of frameworks used for developing unit and integration tests . 100
8.6 Table for Q25, with the number of people for each answer 105

Bibliography

[1] J. Krüger, M. Al-Hajjaji, S. Schulze, G. Saake, and T. Leich, "Towards automated test refactoring for software product lines," in *Proceeedings of the 22nd International Conference on Systems and Software Product Line - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, pp. 143–148, 2018.

[2] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math*, vol. 58, pp. 345–363, 1936.

[3] M. Wahid and A. Almalaise, "Junit framework: An interactive approach for basic unit testing learning in software engineering," in *2011 3rd International Congress on Engineering Education (ICEED)*, pp. 159–164, Dec 2011.

[4] S. Acharya, *Mastering Unit Testing Using Mockito and JUnit*, vol. 62. 2013.

[5] G. A. Meszaros, *xUnit test patterns : refactoring test code*. The Addison-Wesley signature series xUnit test patterns, Addison Wesley, 2010.

[6] V. Menon and C. Nedelcu, *TestNG beginner's guide*. 2011.

[7] F. Appel, *Testing with Junit*. Packt Publishing, 2015.

[8] V. C. Massol and P. C. Tahchiev, *JUnit in Action*, vol. 54. Manning Publications Company, 1995.

[9] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio, "Evaluating advantages of test driven development: A controlled experiment with professionals," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ISESE '06, (New York, NY, USA), pp. 364–371, ACM, 2006.

[10] D. Burns, *Selenium 2 testing tools beginner's guide*. 2010.

[11] a. Garcia, Boni, *Mastering software testing with JUnit 5 :*. Birmingham, England ; Mumbai, [India] :: Packt„ 2017.

[12] O. Taipale, J. Kasurinen, K. Karhu, and K. Smolander, "Trade-off between automated and manual software testing," *International Journal of System Assurance Engineering and Management, 2011, Vol.2(2), pp.114-125*, 2011.

[13] H. K. N. Leung and L. White, "Insights into regression testing [software testing]," in *Proceedings. Conference on Software Maintenance - 1989*, pp. 60–69, Oct 1989.

[14] R. Torkar and S. Mankefors, "A survey on testing and reuse," in *Proceedings 2003 Symposium on Security and Privacy*, pp. 164–173, Nov 2003.

[15] N. M. Minhas, K. Petersen, N. B. Ali, and K. Wnuk, "Regression testing goals - view of practitioners and researchers," in *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, pp. 25–31, Dec 2017.

[16] N. Kolhaupt, *Automated software testin.* 2017.

[17] E. A. Dustin, B. C. Gauf, and T. C. Garrett, *Implementing automated software testing : how to save time and lower costs while raising quality.* Addison Wesley, 2009.

[18] D. J. Mosley and B. A. Posey, *Just enough software test automation /.* 2002.

[19] S. Brutus and G. Greguras, "Self-construals, motivation, and feedback-seeking behaviors," *International Journal of Selection and Assessment, September 2008, Vol.16(3), pp.282-291*, 2008.

[20] B. Hambling, G. Thompson, P. Williams, A. Samaroo, and P. Morgan, *Software testing: an ISTQB-BCS certified tester foundation guide.* BCS Learning & Development, 2015.

[21] Y. Singh, *Software testing.* 2012.

[22] R. Patton, *Software testing /.* 2nd ed. ed., 2006.

[23] N. Verma, *Mobile Test Automation with Appium.* 2017.

[24] A. Bruns, A. Kornstadt, and D. Wichmann, "Web application tests with selenium," *IEEE Software*, vol. 26, pp. 88–91, Sep. 2009.

[25] V. Garousi and J. Zhi, "A survey of software testing practices in canada," *Journal of Systems and Software, May 2013, Vol.86(5), pp.1354-1376*, 2013.

[26] L. A. Franz and J. C. Shih, "Estimating the value of inspections and early testing for software projects," *HEWLETT PACKARD JOURNAL*, vol. 45, pp. 60–60, 1994.

[27] N. Jianyi and Y. Zhengqiu, "An automated test tool of web application based on struts."

[28] C. Mcmahon, "History of a large test automation project using selenium."

[29] M. Slagell, *Sams teach yourself Ruby in 21 days /.* 2002.

[30] a. Amodeo, Enrique, *Learning behavior-driven development with javascript :*. Community Experience Distilled, Birmingham, England :: Packt Publishing Ltd„ 2015.

[31] S. Garg, *Cucumber Cookbook*. Packt Publishing, 2015.

[32] D. Graham and M. Fewster, *Experiences of test automation: case studies of software test automation*. Addison-Wesley Professional, 2012.

[33] W. Afzal, S. Alone, K. Glocksien, and R. Torkar, "Software test process improvement approaches: A systematic literature review and an industrial case study," *Journal of Systems and Software*, vol. 111, pp. 1 – 33, 2016.

[34] C. Garcia, A. Dávila, and M. Pessoa, "Test process models: Systematic literature review," in *International Conference on Software Process Improvement and Capability Determination*, pp. 84–93, Springer, 2014.

[35] R. Vallon, *Lean and Agile Software Development*. 2011.

[36] A. Wintersteiger, *Scrum*. Frankfurt am Main: entwickler.press, 3. aktualisierte und erweiterte auflage ed., 2015.

[37] J. Watkins and S. V. Mills, *Testing IT*. Cambridge [u.a.]: Cambridge Univ. Press, 2. ed. ed., 2010.

[38] N. Gogna, "Study of browser based automated test tools watir and selenium," *International Journal of Information & Education Technology IJIET*, vol. 4, no. 4, 2014.

[39] a. Collin, Mark, *Mastering Selenium WebDriver*. Community Experience Distilled, Birmingham, [England] ; Mumbai, [India] :: Packt Publishing„ 2015.

[40] "Applitools wins 2018 red herring top 100 award," 2018.

[41] R. Narkhede, S. Korde, A. Darda, and S. Sharma, "An industrial research on gui testing techniques for windows based application using uft," in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pp. 466–471, May 2015.

[42] R. Winter, "Agile software development: Principles, patterns, and practices," Apr 2014.

[43] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *arXiv preprint arXiv:1709.08439*, 2017.

[44] P. Ramya, V. Sindhura, and P. V. Sagar, "Testing using selenium web driver," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–7, Feb 2017.

[45] K. Dima, E. Jim, and S. Jeremy, *Selenium design patterns and best practices :*. Community Experience Distilled, Birmingham, England :: Packt Publishing„ 2014.

[46] a. Sams, Prashanth, *Selenium Essentials :*. Community Experience Distilled, Birmingham, England :: Packt Publishing„ first edition. ed., 2015.

[47] A. Contan, C. Dehelean, and L. Miclea, "Test automation pyramid from theory to practice," in *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pp. 1–5, May 2018.

[48] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009.

[49] N. B. Harrison, "Teaching software testing from two viewpoints," *Journal of Computing Sciences in Colleges*, vol. 26, no. 2, pp. 55–62, 2010.

[50] T. Franz and S. V. Wolny, *Automatisiertes White-Box Testen für regelbasierte Modelltransformationen*, vol. 55. 2015.

[51] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Bringing white-box testing to service oriented architectures through a service oriented approach," *The Journal of Systems and Software*, vol. 84, no. 4, pp. 655–668, 2011.

[52] "Unit testing and integration testing," 2013.

[53] O. Cole, "White-box testing," *Dr. Dobb's Journal, Mar 2000, Vol.25(3), pp.23-28*, vol. 30, pp. 81–87, 2014.

[54] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: empirical observations," *Advances in Software Engineering*, vol. 2010, 2010.

[55] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.

[56] F. Besson, P. Moura, F. Kon, and D. Milojicic, "Bringing test-driven development to web service choreographies," *Journal of Systems and Software*, vol. 99, pp. 135 – 154, 2015.

[57] a. Farcic, Viktor and a. Garcia, Alex, *Test-driven java development : Invoke TDD principles for end-to-end application development*. Community Experience Distilled, Birmingham, England ; Mumbai, India :: Packt Publishing„ 2018.

[58] V. Garousi, M. Felderer, and T. Hacaloğlu, "Software test maturity assessment and test process improvement: A multivocal literature review," *Information and Software Technology*, vol. 85, pp. 16 – 42, 2017.

[59] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen, "Reversible debugging software," *Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep*, 2013.

[60] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Berlin, Heidelberg: Springer Berlin Heidelberg, 2012 ed., 2012.

[61] T. Ternité, "Process lines: A product line approach designed for process model development," in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 173–180, Aug 2009.

[62] M. Becker and B. Zhang, "How do our neighbours do product line engineering?: a comparison of hardware and software product line engineering approaches from an industrial perspective," in *Proceeedings of the 22nd International Conference on Systems and Software Product Line - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, pp. 190–195, 2018.

[63] K. Hayashi and M. Aoyama, "A multiple product line development method based on variability structure analysis," in *Proceeedings of the 22nd International Conference on Systems and Software Product Line - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, pp. 160–169, 2018.

[64] J. Link, *Unit Testing in Java*. Morgan Kaufmann, 2003.

[65] J. Simmonds, D. Perovich, M. C. Bastarrica, and L. Silvestre, "A megamodel for software process line modeling and evolution," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 406–415, Sept 2015.

[66] M. Löwisch, G. Caspers, and S. Klumpp, *Arbeitsrecht*. Verlag Franz Vahlen GmbH, 2017.

[67] H. Washizaki, "Building software process line architectures from bottom up," in *International Conference on Product Focused Software Process Improvement*, pp. 415–421, Springer, 2006.

[68] D. Rombach, "Integrated software process and product lines," in *Software Process Workshop*, pp. 83–90, Springer, 2005.

[69] O. Armbrust, M. Katahira, Y. Miyamoto, J. Münch, H. Nakao, and A. Ocampo, "Scoping software process lines," *Software Process: Improvement and Practice*, vol. 14, no. 3, pp. 181–197, 2009.

[70] H. Royer, Jean-Claude;Arboleda, *Model-Driven and Software Product Line Engineering*. Wiley-ISTE, 2013.

[71] A. Fuggetta and E. Di Nitto, "Software process," in *Proceedings of the on Future of Software Engineering*, FOSE 2014, (New York, NY, USA), pp. 1–12, ACM, 2014.

[72] M. C. Paulk, "Comparing iso 9001 and the capability maturity model for software," *Software Quality Journal*, vol. 2, no. 4, pp. 245–256, 1993.

[73] C. P. Team, "Cmmi for services, version 1.3," *CMU SEI, Nov-2010*, 2010.

[74] S. Wartanian, *Modernization of the software-engineering-process within a large project-environment*. 2016.

[75] M. C. Paulk, "How iso 9001 compares with the cmm," *IEEE Software*, vol. 12, pp. 74–83, Jan 1995.

[76] M. C. Paulk, "Comparing iso 9001 and the capability maturity model for software," *Software Quality Journal*, vol. 2, pp. 245–256, Dec 1993.

[77] M. a. Hinsch, *Die neue ISO 9001:2015 - Status, Neuerungen und Perspektiven*. Berlin, Heidelberg :: Springer Berlin Heidelberg : Imprint: Springer Vieweg„ 2015.

[78] M. Sokovic, D. Pavletic, and K. K. Pipan, "Quality improvement methodologies–pdca cycle, radar matrix, dmaic and dfss," *Journal of achievements in materials and manufacturing engineering*, vol. 43, no. 1, pp. 476–483, 2010.

[79] A. L. Mesquida, A. Mas, E. Amengual, and J. A. Calvo-Manzano, "It service management process improvement based on iso/iec 15504: A systematic review," *Information and Software Technology*, vol. 54, no. 3, pp. 239 – 247, 2012.

[80] S. Peldzius and S. Ragaisis, "Comparison of maturity levels in cmmi-dev and iso/iec 15504," *Applications of Mathematics and Computer Engineering*, pp. 117–122, 2011.

[81] S. Hwang and H. Yeom, "Analysis of relationship among iso/iec 15504, cmmi and k-model," in *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pp. 306–309, May 2009.

[82] SIG-Automotive, "Automotive spice-process assessment model (2007)," *Website: http://www. automotivespice. com/fileadmin/softwaredownload/automotiveSIG_PAM_v25. pdf (Stand 01.09. 2014)*, 2017.

[83] H. Washizaki, "Building software process line architectures from bottom up," in *Product-Focused Software Process Improvement* (J. Münch and M. Vierimaa, eds.), (Berlin, Heidelberg), pp. 415–421, Springer Berlin Heidelberg, 2006.

[84] V. Garousi, M. Felderer, and T. Hacaloğlu, "What we know about software test maturity and test process improvement," *IEEE Software*, vol. 35, pp. 84–92, January 2018.

[85] N. King, C. Horrocks, and J. Brooks, *Interviews in qualitative research*. SAGE Publications Limited, 2018.

[86] S. Kvale and S. Brinkmann, *InterViews: Learning the Craft of Qualitative Research Interviewing*. SAGE, 2009.

[87] F. J. Fowler Jr, *Survey research methods*. Sage publications, 2013.

[88] F. T. W. Au, S. Baker, I. Warren, and G. Dobbie, "Automated usability testing framework," in *Proceedings of the Ninth Conference on Australasian User Interface - Volume 76*, AUIC '08, (Darlinghurst, Australia), pp. 55–64, Australian Computer Society, Inc., 2008.

[89] B. DiCicco-Bloom and B. F. Crabtree, "The qualitative research interview," *Medical education*, vol. 40, no. 4, pp. 314–321, 2006.

[90] T. Dyba and T. Dingsoyr, "What do we know about agile software development?," *IEEE Software*, vol. 26, pp. 6–9, Sep. 2009.

[91] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Berlin, Heidelberg: Springer Berlin Heidelberg, 2014 ed., 2014.

[92] D. Hauser, *Test-driven iOS development with Swift*. Community experience distilled, Birmingham: Packt Publishing, 2016.

[93] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on visual gui testing," *Information and Software Technology*, vol. 73, pp. 66–80, 2016.

[94] S. Sivanandan and C. B. Yogeesha, "Agile development cycle: Approach to design an effective model based testing with behaviour driven automation framework," 2014.

[95] L. Gonçalves, "Scrum," *Controlling and Management Review, 2018, Vol.62(4), pp.40-42*, 2018.

[96] T. Epping, *Kanban für die Softwareentwicklung*. Informatik im Fokus, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[97] T. Björkholm and J. Björkholm, *Kanban in 30 days*. Birmingham: Impackt Publishing, 2015.

[98] R. Lopez-Herrejon, S. Illescas, and A. Egyed, "A systematic mapping study of information visualization for software product line engineering," *Journal of Software: Evolution and Process, February 2018, Vol.30(2)*, 2018.

[99] D. Dias de Carvalho, L. F. Chagas, and C. A. L. Reis, "Definition of software process lines for integration of scrum and cmmi," in *2014 XL Latin American Computing Conference (CLEI)*, pp. 1–12, Sep. 2014.

[100] a. Martin, Robert C. and R. C. M. s. Martin, Robert C., *Clean architecture :*. Robert C. Martin series, Boston :: Prentice Hall„ 2018.

[101] M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro, "Improving test suites maintainability with the page object pattern: An industrial case study," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 108–113, March 2013.

[102] P. H. Ruiz, C. Camacho, and J. A. Hurtado, "A comparative study for scoping a software process line," in *2018 ICAI Workshops (ICAIW)*, pp. 1–6, Nov 2018.

Online References

[103] "Schibsted website structure." https://schibsted.com/. Accessed: 2018-06-28.

[104] "Calabash ios." https://github.com/calabash/calabash-ios. Accessed: 2018-07-04.

[105] "Calabash android." https://github.com/calabash/calabash-android. Accessed: 2018-07-04.

[106] "Calabash." http://calaba.sh/. Accessed: 2018-07-05.

[107] "Calabash framework definition." https://bitbar.com/calabash-tutorial-for-mobile-app-testing/. Accessed: 2018-07-05.

[108] "Mobile device emulator and simulator vs real device." https://saucelabs.com/blog/mobile-device-emulator-and-simulator-vs-real-device. Accessed: 2018-07-05.

[109] "Watir framework." http://watir.com/. Accessed: 2018-08-20.

[110] "Watir framework on github." https://github.com/watir/watir. Accessed: 2018-08-21.

[111] "Watir vs. selenium." https://dzone.com/articles/selenium-vs-watir. Accessed: 2018-08-21.

[112] "Ranorex homepage." https://www.ranorex.com/. Accessed: 2018-08-21.

[113] "Ranorex webtestit." https://www.ranorex.com/webtestit/beta/. Accessed: 2018-08-25.

[114] "Selenium hq." https://www.seleniumhq.org/. Accessed: 2018-08-26.

[115] "Geckodriver firefox." https://github.com/mozilla/geckodriver/. Accessed: 2018-08-30.

[116] "Chromedriver." http://chromedriver.chromium.org/. Accessed: 2018-08-30.

[117] "Issue for slenium ide." https://github.com/SeleniumHQ/selenium/issues/4406. Accessed: 2018-08-30.

[118] "Introduction to selenium ide." https://www.guru99.com/introduction-selenuim-ide.html. Accessed: 2018-08-31.

[119] "Exporting tests from selenium ide." http://elementalselenium.com/tips/6-export-from-selenium-ide. Accessed: 2018-09-02.

[120] "Testng vs junit: What's the difference?." https://www.guru99.com/junit-vs-testng.html. Accessed: 2018-09-27.

[121] "Junit 5." https://junit.org/junit5/. Accessed: 2018-09-28.

[122] "Kotlin." https://kotlinlang.org/docs/reference/faq.html#who-develops-kotlin. Accessed: 2018-11-15.

[123] "Swift." https://itunes.apple.com/gb/book/swift-programming-language/id881256329. Accessed: 2018-11-15.

Survey Questions for

Improving Test Automation BestPractices with Test Process Lines

A Case Study on Classified Ads Platforms

General questions:

1. What is your name? (Optional)
2. For which company are you working?
3. For which department are you working?
4. What is your role in this company?
5. How many years are you working in that position (role)?
6. How much overall experience do you have in your field of work (e.g. in software development)?

Project management & process management questions:

7. Does the team you are part of operate under a certain software development process?
8. What is the most used software development process in your context?
9. Is software testing a part of the project planning cycle?
10. Which software tests types are part of the software development life cycle?
11. What is the share of software testing related effort to the software development effort?
12. What is the main focus of software testing?

Software testing questions:

13. Are you familiar with software testing?
14. Which language are you using for your projects?

    Others, please specify:

    If you are developing a Unit/Integration test case in a different language, please specify:
15. Which framework are you using for your Unit/Integration tests?

    Others, please specify:
16. How satisfied are you with the amount of testing that is done in your company?

    Please give some comment on your statement:

Manual testing questions:

17. Does your company have explicit manual software tester?
18. Are manual testers' part of the development teams?
19. Do you think that manual testing is required in your projects?
20. Why do you think so?
21. Should more manual testing be done in your company?

    Please explain your statement:
22. How often do you execute the manual tests?
23. What is the scope of manual testing?
24. How do you manage software tests (test plan, test report, test recommendation)?

Automated End-To-End (E2E) Tests questions:

25. Are E2E tests or functional tests used in your company?
26. Which language are you using for these tests?

    Others, please specify:
27. Who is responsible for writing End-To-End tests?
28. Which frameworks are you using for your End-To-End Tests

    Others, please specify:

Survey Completion:

29. Any further comments? (e.g. Improvements,...)
30. May we contact you to discuss specific topics via E-Mail/phone/skype?
31. Do you want to receive the survey results via email?

Name: (Optional)

E-Mail address: (Optional)