CHALMERS

Structuring GUI Acceptance Tests based on Usage Scenarios

Master of Science Thesis in Software Engineering and Technology

GRISCHA LIEBEL

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, February 2013

Structuring GUI Acceptance Tests based on Usage Scenarios

Grischa Liebel

Examiner: Christian Berger

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, February 2013

Note

This thesis report was written in connection with a Master's thesis project as an external student at Chalmers University of Technology, Gothenburg, Sweden. As the author was not enrolled in any Master's programme at Chalmers University of Technology, the project lead to no academic degree there. However, the thesis report was handed in for grading at Karlsruhe Institute of Technology, Karlsruhe, Germany, where the author was enrolled in a graduate programme at the time of writing. Therefore, it is also published by Karlsruhe University of Technology under the same title.

Acknowledgements

I would like to express my gratitude to Robert Feldt, Chalmers University of Technology, for supervision, support, and guidance during the planning phase and throughout the whole thesis project. Additionally, I would like to thank Emil Alégroth, Chalmers University of Technology, for supervising the thesis and providing valuable input throughout the thesis project. Thanks go to Ralf Reussner, Karlsruhe Institute of Technology, for being open for the idea of an external thesis project, supporting it, and supervising it at Karlsruhe Institute of Technology.
A special *Thank you* goes to all the contact persons and interviewees at the different participating companies. I am thankful for the valuable time and trust you offered me.
A very special thanks to Anna Kryvenda for ongoing support and encouragement throughout the thesis project. I really appreciate that, ma'am!
Finally, I would like to thank the Swedish weather for leaving no doubt that my place is inside at the desk!

Abstract

Software testing plays an important role in the overall software development process. However, many areas of software testing are still unexplored, or contain gaps in their existing body of knowledge, such as the area of GUI acceptance and system testing. This thesis aims to bridge this gap with results from a multiple-case study with the goal of investigating the current state of GUI acceptance and system testing in industry. The study was performed with six software development companies, of different size and context, to acquire broad information and a general overview of industrial state-of-practice. From the acquired information, the PASTA framework on how to approach GUI acceptance and system testing has been derived. The framework contains guidelines for two approaches to GUI acceptance and system testing. Firstly, an organised-level testing approach and secondly a base-level testing approach. Furthermore, these guidelines represent a holistic approach to this type of testing, with practices for test suite design, test scenario elicitation, test automation, and test maintenance. Finally, the thesis presents how these practices were evaluated with the participating companies. Results of the thesis show that the framework can increase the benefits of GUI acceptance and system testing. Consequently, the results of this work help to bridge the current gap in knowledge and constitute a valuable contribution to industry.

Contents

1. Introduction

In the past, defects in software systems have often lead to critical failures. Among these, the explosion of the Ariane 5 rocket on its maiden flight in 1995 is probably the most famous [37]. Even if the defects are not critical, they can lead to increased development times, higher costs in the development and maintenance phases, or even to the failure of whole software projects. Software tests are a widely-accepted practice to find errors in the software system early in the engineering process and therefore to reduce the number of errors in the resulting software system.
Software tests can be sorted into many different categories. One way to do so which is commonly used is to divide them into four different categories depending on the system level on which they are used [20]. *Unit* tests focus on the code level and test single functions or methods. *Integration* tests focus on the integration between parts of the system like classes, components, or modules. *System* and *acceptance* tests are performed on the complete system and aim at validating if the specifications and requirements have been fulfilled. Acceptance testing is defined as *"Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component."* [1]. System testing, on the other hand, is defined as *"Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements."* [1]. That is, acceptance and system testing differ mostly with regard to the input data and the unit which is conducting the tests.
Conducting system and acceptance tests, and testing in general, is costly. Studies show that over "fifty percent of the cost of software development is devoted to testing" [27]. By automating manual tests, these costs could be reduced. However, it is often stated that automated tests can not replace manual tests completely [12]. Instead, many approaches focus on how to prioritise test cases so that by only executing the most important tests, the testing costs can be reduced [22]. Additionally, existing approaches on automated testing on system level suffer from limitations [16].

As graphical user interfaces, or GUIs, are becoming ubiquitous in software systems, testing needs to be done on GUI level as well. In contrast to classical software testing, GUI testing has to be approached in a different way [42]. For instance, it is stated that traditional test coverage criteria are not useful for GUI testing [42]. The *test coverage* of a system or a software product is hereby defined as the *"extent to which the test cases test the requirements for the system or software product."* [1].

While all these areas play an important role in research, there are comparably few approaches focusing on the area of GUI acceptance and system testing as one. Existing approaches are either unsuited for GUI testing on system level or are associated with an extensive time overhead. Additionally, more empirical research on current testing practices in industry is needed [40].

1.1. Aim

This thesis project aims at developing a framework on how to conduct GUI acceptance and system testing based on an empirical study on current testing practices in industry. The framework is named Planned Acceptance and System Testing Adoption framework, or PASTA framework. PASTA will help with employing GUI acceptance and system testing within projects in a planned way.
In order to reach this goal, the current state in GUI acceptance and system testing will be studied in several software-developing teams. The information acquired in industry, together with current research in different areas of system testing, acceptance testing, GUI testing, and in scenario usage, will aid in identifying advantages and problems with current testing approaches and in developing the framework.
Finally, the PASTA framework will be evaluated statically together with the studied companies in order to find flaws in it and refine it. The research methodology is discussed in detail in chapter 3.

1.2. Research Questions

In the course of this thesis, the author tries to answer the following research questions:

- RQ1: How is GUI acceptance and system testing done in practice?
- RQ2: Are there best practices or methodologies in other areas of software engineering which can be directly transferred to GUI acceptance and system testing?
- RQ3: Can these practices or methodologies be merged into a single framework for GUI acceptance and system testing?
- RQ4: Can the resulting framework facilitate GUI acceptance and system test creation, lead to a higher grade of automation, and reduce the maintenance effort of the test suites?

The first question is focused on assessing the current state-of-practice in industry. This research is not aimed to be on a purely academic level but rather to be adapted to the needs of the software developing industry. Even though software engineering is considered an applied science, empirical evaluations are often missing. Tichy and Höfer state that in empirical software research “Important topics are underrepresented or absent” [31]. Sjøberg et al state that there are “relatively few empirical studies.” and that “The use of empirical methods by the software industry is low.” [55]. By conducting a multiple-case study in a number of industry projects at six software-developing companies, the author tries bridge the gap between research and practice.
The second question aims at transferring the existing knowledge from research in other fields of software engineering to the design of GUI acceptance and system tests. Test scenario elicitation is related to requirements elicitation, as the tests aim at verifying requirements and as requirements can be expressed in testable scenarios. Test suite design can be seen as related to the design of software systems, especially if the test suite is automated. Similarly, test automation and test maintenance have commonalities with software development in general. By comparing existing practices from these more mature fields of

software engineering with the information acquired in industry, commonalities among the two areas are determined. This could help in finding solutions for GUI acceptance and system testing.
After having gathered information from real-life projects through the study, the author tries to abstract from these projects to a more general level. This abstraction is especially relevant as all the preceding work steps are project-focused. Project-focused guidelines would prevent the developed framework from being adaptable by other projects and would therefore threaten the external validity of the project.
As a last step, it is answered if the developed framework can improve the current state in industry and solve existing problems with GUI acceptance and system testing. The PASTA framework is therefore discussed together with the teams studied during the case study.

1.3. Contributions

The contributions of this thesis to existing research are as follows:

- The assessment of the state-of-practice in GUI acceptance and system testing based on an exploratory multiple-case study at a number of different teams at six companies.
- The assessment of typical problems and solutions in the areas of test creation, test execution, and the usage of tools for GUI acceptance and system tests.
- The introduction of a multi-level framework, the PASTA framework, on how to approach GUI acceptance and system testing.
- Guidelines on how to implement each level of the PASTA framework.
- The static evaluation of the proposed framework with the participating teams.

1.4. Limitations

There are a number on limitations in this thesis project.
Firstly, the number of studied projects for information gathering is limited due to time constraints. Therefore, the gathered information can only pose a small set of practices and experiences in GUI acceptance and system testing. Also, the projects are selected by means of time, interest, and personal contacts. That is, only companies which could spare enough time to participate in the case study, which were interested in the project, or to which personal contacts existed were included in the case study. Selection bias can therefore not be ruled out. Furthermore, the means of data collection are restricted to interviews and group sessions in the studied teams.
Secondly, the PASTA framework is not intended to be complete nor is it intended to formalise the test creation or execution process. The author believes that at the current state of GUI testing in practice and research, too much formalisation would hinder the adaptation of any framework.
Thirdly, automation of GUI test creation or execution is not a primary aim of this thesis. The PASTA framework is intended as a starting point which could lead to a higher degree of automation in the future. However, more research will be needed in order to reach this aim.
Finally, again due to time constraints, only a statical evaluation could be performed. That is, the PASTA framework was discussed with the teams and a survey was answered by every interviewee. Therefore, it might not be generalisable to every project context.

1.5. Structure

The remaining part of this thesis is structured as follows: Chapter 2 will introduce relevant research related to the thesis topic. Publications out of the area of system testing, acceptance testing, GUI testing, and scenario usage will be discussed. In Chapter 3, the used research methodology is described in detail. The theoretical framework underlying the research is described and justified. The chapter is followed by the analysis of the acquired data together with the design of the PASTA framework in Chapter 4. The static evaluation of the PASTA framework is covered by Chapter 5. Both the design of the evaluation and the results will be discussed here. The validity of the data collection, data analysis, and evaluation will be discussed in chapter 6. The thesis is summarised and possibilities for future work are discussed in Chapter 7.

2. Related work

Software tests are often divided into the four different categories of *unit*, *integration*, *system*, and *acceptance* testing. This thesis focuses on GUI acceptance and system testing on GUI level. That is, an entire system is tested through its GUI. There are numerous approaches in acceptance testing, system testing and GUI testing. However, few focus on the combination of GUI testing and acceptance or system testing. Therefore, the following sections will introduce the areas of research separately.
Firstly, current practice and past research in system and acceptance testing will be presented. Secondly, the use of scenarios in testing is explained. As test cases are usually derived from scenarios, they play an important part in the overall testing process. Thirdly, existing GUI testing approaches will be discussed. Finally, it is outlined where this thesis fits in and which gaps exist in current knowledge.

2.1. System and acceptance testing

System testing is defined as *"Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements."* [1]. Many approaches in system testing are concerned with using models to automatically derive system test scenarios. According to [34], the "generation of test cases at an early stage of the software development process so that testing (test case generation) and coding activities are carried out in parallel" is one of two main challenges arising from shorter software developing cycles. In [47], a modified version of UML use case diagrams is used to create so-called *test objectives*. These are basically system test cases. The test objectives can then be used for manual testing. Additionally, sequence diagrams can be provided in order to generate running tests. Use-case scenarios written in a restricted natural language format are used to generate tests on system level in [57]. Single use cases are transferred to a control-flow diagram. The individual diagrams are merged to a global control-flow diagram. Coverage criteria are used to derive test cases from this global diagram. There is a substantial overhead for diagram creation and modification of the use case description to the restricted natural language format in this approach. Briand and Labiche introduce the TOTEM methodology to derive system test requirements, test cases, and further artefacts from models created in the analysis phase of a project [14]. The approach requires extensive UML modelling of the system including OCL annotations. In the context of embedded, object oriented software, Nebut et al derive test cases from documented requirements [48]. Again, UML use case diagrams are used. The work is empirically evaluated in three case

studies.
Another common area of research regarding system testing is the prioritisation of test cases. The need for this kind of research arises through the fact that extensive system testing is costly and often impossible. This is especially the fact for regression testing, where tests are repeated regularly. Kundu et al state this as the second main challenge arising from shorter software development cycles: "focus on the testing of those parts in the code, which are of higher priority in view of criticality or complexity so that better quality software is developed with a limited testing effort" [34]. Apart from stating these two challenges, they develop the STOOP framework for generating test scenarios from UML sequence diagrams, prioritising the test cases based on three different metrics, and generating test data according to the prioritisation in [34]. An empirical study on test prioritisation techniques with regard to regression testing is presented in [22]. The study shows that prioritisation techniques can significantly increase the fault detection rate of a test suite in comparison to a randomly selected test suite. An approach to prioritise system test cases based on four factors regarding the requirements they test is presented in [58]. Three industrial studies are conducted using the proposed approach. These are presented in [58] as well. Especially the priority a customer assigns to each requirement proved to play an important role in system test prioritisation. The studies show similar support for test prioritisation in general as the studies conducted in [22].

Acceptance testing is defined as *"Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component."* [1]. It is furthermore often stated that acceptance tests are supposed to be written by the customer [52]. As acceptance and system testing both operate on system-level, a lot of techniques from system testing can be applied here as well. The main focus in research therefore lies on presenting the person writing the tests with an easy-to-use interface and syntax for defining scenarios in order to enable customers to write the tests. These scenarios are then automatically translated to test plans or directly to executable tests. A popular tool in this area is FIT [46] and a multitude of specialised tools building on the FIT framework, like FitNesse for testing web applications [2]. FIT presents the user with a tabular format to define test cases which are then translated to running tests using so-called *fixtures*. These fixtures have to be implemented by the developers. An industrial case study by Haugset and Hannssen states both positive and negative effects that arise in industry projects using FIT [28]. It is reported that developers get a better overview of the project's status and the actual function of the software system. However, in most projects which were investigated, the tests were still mostly written by the developers, sometimes leading to a high test-creation overhead. Also, the authors report potential dangers with fixtures becoming too complex when the tests are written up-front in a test-first way. Practical experience with FIT projects is collected and presented, among other contributions, in [41]. It is stated that the main challenge with executable automated acceptance tests is maintaining the tests. It is also suggested that acceptance tests should mainly be done at the layer directly under the graphical user interface [41]. However, the authors do not address the possibility that there might be errors in the graphical representation even though the layer under the GUI works as expected. In this case, the acceptance test done on the layer under the GUI would yield false positives.
A common approach to derive acceptance tests is, similar to the research on system testing, by using UML diagrams. Heinecke et al. generate test plans for acceptance tests from UML activity diagrams [29]. The assumption is that business processes are modelled using UML anyway. Therefore, the diagrams can directly be used for test plan creation. The test plans themselves are not executable, though, and need to be implemented or executed manually. The authors implemented only a prototype of the test plan generator, so extensive practical experience using their approach is missing. An approach focusing

on web applications is presented by [9]. Here, so-called use case maps are used to derive acceptance tests. The authors state that lightweight, scenario-based methods are better for web developers. The tests are executed automatically using the FitNesse framework. Therefore, the same overhead for fixture creation exists as in all FIT-related methods. In [52], the main differences between unit testing and acceptance testing are explained from the perspective of a developer. The author states that acceptance testing is a practice which is "hard to get going", mainly because of customer involvement. Furthermore, it is mentioned that programming languages are unsuited for writing automated acceptance tests as the customer does not understand them, natural languages are too verbose to define tests, and script languages can be too complicated for the customer to use them.

2.2. Scenario usage

A scenario is *"A step-by-step description of a series of events that may occur concurrently or sequentially"* [1]. Scenarios are widely used in requirements engineering, human-computer interaction and strategic management [33]. They are mainly being used as a means to stimulate thinking and "to improve communication between developers and users." [33]. Becks et al. use textual analysis and semantic maps for structuring scenarios [11]. This is mainly undertaken to synchronise knowledge collected by different or distributed teams. A strategy for managing scenarios is presented in [8]. The approach uses a similarity measure between scenarios which depends on so-called *episodes*, that is "a named subsequence usually shared among several scenarios" [8]. A potential tool is discussed, but no prototype or implementation presented. A broader view on scenario usage and scenario management is presented in [33]. It is especially interesting as it does not only restrict itself on Software Engineering and might therefore be of particular importance for acceptance testing. As customers will regularly have different backgrounds, such as management, it is important to know how scenarios are used in their areas of profession. Scenarios are also widely used in software testing. Cunning et al. present a way to generate test scenarios from semi-formal requirements scenarios [21]. The authors claim that for testing, a semi-formal approach is needed as it is both not too vague and not too complicated to be adopted. However, the requirements need to be written in the presented format which means that applying the approach to existing projects is difficult. Natural language scenarios from requirements engineering are translated to annotated state charts in [54]. Furthermore, a step-by-step procedure for eliciting scenarios is presented. These scenarios are then used for system testing. The authors use so-called *dependency charts* to depict dependencies between scenarios. The approach is kept on a theoretic level giving only some examples of modelling. Tsai et al present a framework for testing of open source software [60]. The framework takes scenarios as an input which can then be executed on the system on different levels. A lot of scenario-based approaches derive test cases from UML models. To these, [47, 57, 34, 14, 29] can be counted as well. Additionally, experience with deriving test cases from UML sequence diagrams in the area of healthcare and employability is reported in [39].

2.3. GUI Testing

GUI testing is reported to be challenging [35]. In his article on GUI testing [42], Atif M. Memon argues that GUI testing is different from what he calls *conventional testing*. He states that coverage criteria are not necessarily useful for GUI testing and that a one-step-at-a-time execution is important to trace errors. An empirical study by Memon and Xie on design principles for GUI test suites is presented in [62]. The authors claim that existing testing tools do not support GUI testing. Furthermore, "Current techniques force testers to test GUIs on a per test case basis.". The study focuses on the effects of test suite

size, test case length and test case composition. Common industry problems with GUI acceptance testing are presented in [38]. It is stated that writing and maintaining tests has to be easy, otherwise it will not be done. Also, the authors claim that test scripts must be centred around common GUI concepts like clicking a button or entering text. Interestingly, the authors experienced that GUI testing on unit level was increasingly omitted when extensive GUI acceptance testing was used.
Manually, testing a system through its GUI is often done by releasing beta versions to the product's customers [42]. Additionally, exploratory testing can be used to manually test a system through its user interface [59]. A method to break down the GUI into two tiers - a component and a system tier - is presented in [35]. This leads to two tiers that can be tested independently with methods similar to conventional testing. An approach to derive test cases for GUI testing based on so-called *complete interaction sequences (CIS)* is presented in [61]
An attempt to connect conventional automated testing with GUI testing is made in [45]. Therefore, the test data itself is separated from the GUI representation. However, numerous different text and script files are needed which make the approach highly complex. Furthermore, no practical experience with the tool is reported.
In [15], so-called *usage profiles* are generated from usage data collected on a fielded version of a software system. These profiles are then used to create a probabilistic usage model of the software on an abstract GUI level. To avoid breaking the tests due to evolution of the GUI, an abstract model of the GUI is created. Furthermore, a fielded version of the software system is needed. The main drawback of the method is that the used tool needs to be able to access the usage data in some way. This means that the tool will depend on the operating system and the programming language being used.
In [23] and [24] a white-box testing method for graphical user interfaces using the Standard Widget Toolkit [5] is presented. As this approach focuses on unit testing using branch coverage as a measure for the quality of the test suite, it is not directly relevant for this thesis. However, the authors introduce an abstract formalisation for graphical user interfaces in general which could also be applied for representing acceptance tests.
Memon et al. [43] describe test cases as *tasks* and model them in a so-called *event-flow graph*. The approach is used in conjunction with planning algorithms to enable automated regression testing of GUIs. Especially the abstract concept of tasks is relevant with respect to acceptance testing and scenario-based approaches.
Paiva et al. [50] use UML for creating complex GUI models. Using them, they automatically generate test cases. In [26] their tool is furthermore extended to create models from GUIs in a semi-automatic fashion. The drawback is that the approach is only applicable to Windows programs using the .NET framework.

In the area of automated GUI testing, so-called *Capture and Replay* tools, such as the open source web browser automation tool Selenium [3], are widely used due to their ease of use. These tools provide means to record interactions with the GUI and replay them later on. The interactions are either recorded in a white-box fashion, for instance through byte code instrumentation, or simply by recording input actions. However, even small changes in the graphical user interface can break the tests. Therefore, it is often stated that a high maintenance effort is required to keep the tests running, making them unsuited for regression testing. Additionally, the have been stated to be "frustrating to work with" [17]. White-box testing techniques require access to the source code and are therefore usually restricted to single programming languages.
In order to avoid the inherent problems with Capture and Replay tools, Chang et al. present their tool Sikuli in [17]. They use computer vision to detect interaction with the GUI instead of recording coordinates or registering the events on API level. Therefore, the tests are not as sensitive to changes in the graphical user interfaces as tests implemented

using Capture and Replay tools. The authors state that as long as GUI design evolves incrementally, most test cases can be reused in newer versions of the system under test. Börjesson and Feldt name this approach *Visual GUI Testing (VGT)* [16]. A study at Saab AB conducted by the authors shows initial support that automated system testing using VGT could be used in industry. They compare Sikuli to a commercial Visual GUI Testing tool and to manual testing, which showed differences between the tools that make them more beneficial in different contexts. Furthermore, the study showed that VGT scripts have several potential benefits compared to testing with equivalent manual test cases.

2.4. Knowledge gap

The research on system testing is mostly focused on automatically deriving test cases, prioritising test cases, or on tools which facilitate acceptance testing for customers. Even though the defect detection rate can be increased using test case prioritisation, system test suites are still limited if parts of the test cases are not executed. By automating the tests, this could be avoided. However, it is often stated that automated testing can never fully replace manual testing [12], [32]. Existing techniques for automatically creating test scenarios and sometimes even execute them are costly and often require an extensive modelling overhead and expert tool knowledge. This applies to many scenario-based testing methods as well which require a high overhead for model creation or assume that the models already exist.
GUI testing is seen as challenging [35]. Some publications even state that doing no GUI testing at all is most popular in industry [44]. Common ways to test a system manually through its GUI are beta tests [42] and exploratory testing [59]. However, they suffer from the same drawbacks as manual system testing in general. For automated GUI testing, *Capture and Replay* tools can be used. However, they often lead to frustration and the work with them is reported to be slow [17]. Initial support for system testing using visual GUI testing tools is shown in [16]. They could also enable customers to easily create automatic GUI acceptance tests without a deep technical knowledge. However, test cases still have to be created and maintained one by one in a manual way.
Therefore, there is more research required on both manual and automatic GUI testing techniques in general. Especially studies which do not focus on one detail of GUI acceptance and system testing but cover the whole testing process seem to be under-represented in research. Hereby, the current practice in industry is relevant and should be studied in more detail. As stated by Martin et al, “Software Engineering’s programme of empirical research on testing can benefit from studies of work that seek to understand testing as it happens.” [40].

This project therefore focuses on deriving a framework for structuring GUI acceptance and system test suites and for creating the single tests based on the current state of practice in industry. The aim is to make the whole test creation process more structured and test artefacts reusable. Furthermore, a defined structure could facilitate automation for instance in test case generation in future work. Using this framework should lead to GUI test suites which are easily maintainable, understandable and executable. Additionally, it should guide practitioners in the design of their GUI acceptance and system test suite.

3. Research methodology

The research for this thesis is divided into three steps. Firstly, a multiple-case study is conducted in industry to reveal the current state of practice in GUI acceptance and system testing. The cases are software developing teams within different companies. Secondly, from the data gathered in the study, and the current state of research described in chapter 2, the PASTA framework is derived. It focuses on how to approach GUI acceptance and system testing. It consists of two possible test suite levels, factors indicating those levels, and guidelines in the areas of test suite design, test scenario elicitation, test implementation, and test maintenance. Finally, the PASTA framework is evaluated with the companies which participated in the case study conducted in the first step. The theoretical framework for this three-step research will be explained in the following sections. It involves hypotheses made about the research, case study questions which need to be answered in order to support or contradict the hypotheses, and variables within the field that are considered together with their relationships. This theoretical framework will addressed regularly during all following chapters. A detailed overview over the three-step research methodology is given in figure 3. The parts it consists of will be addressed in the following sections and in chapters 4 and 5.
The different kinds of data collected and the techniques used for data collection are explained in the section following the theoretical framework. The case study context is explained along with it as it influences the types of data which were considered during the preparation of the data collection phase.
Section 3.3 covers the analysis of the collected data. It is described how the data is categorised and presented, and how conclusions are drawn from it. The connection to the theoretical framework is made clear.

3.1. Theoretical framework

The multiple-case study conducted as the first step of this thesis is exploratory in nature. That is, its aim is "finding out what is happening, seeking new insights and generating ideas and hypotheses for new research" [53]. The study aims at determining the current state in GUI acceptance and system testing and therefore answering the first research question, "How is GUI acceptance and system testing done in practice?". As a secondary aim, the study is also intended to answer the second research question, "Are there best practices or methodologies in other areas of software engineering which can be directly transferred to GUI acceptance and system testing?", together with additional input from

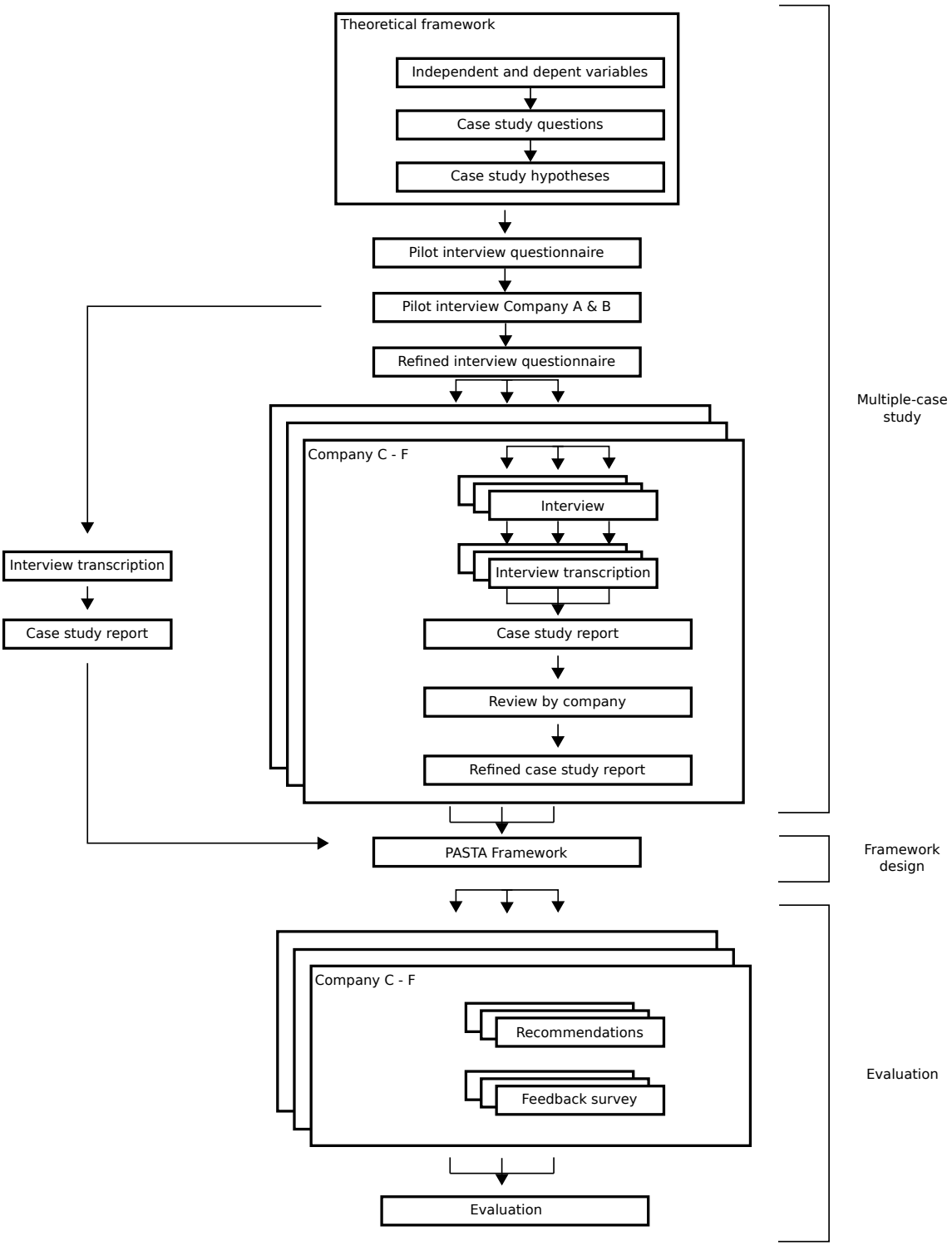

Figure 3.1.: Overall research methodology

research.
As outlined in section 2, there is few directly related work on the topic of GUI acceptance and system testing. Therefore, using an exploratory case study design is justified in order to understand the current state-of-practice.
Firstly, a number of *independent variables* are considered. These are the main influence factors to research questions RQ1 and RQ2 and the area of GUI acceptance and system testing in general according to the author's understanding of the area. Also, these variables help keeping the research on track in the following research phases. Secondly, a number of *case study questions* are presented. These need to be answered in order to answer the overall research questions. Finally, the theoretical framework consists of *hypotheses* related to the case study questions. These hypotheses are made based on existing literature in other related areas of research.
The independent variables used in the theoretical framework are depicted in figure 3.1. They all influence the dependent variable, namely *Current practice in GUI acceptance and system testing.*

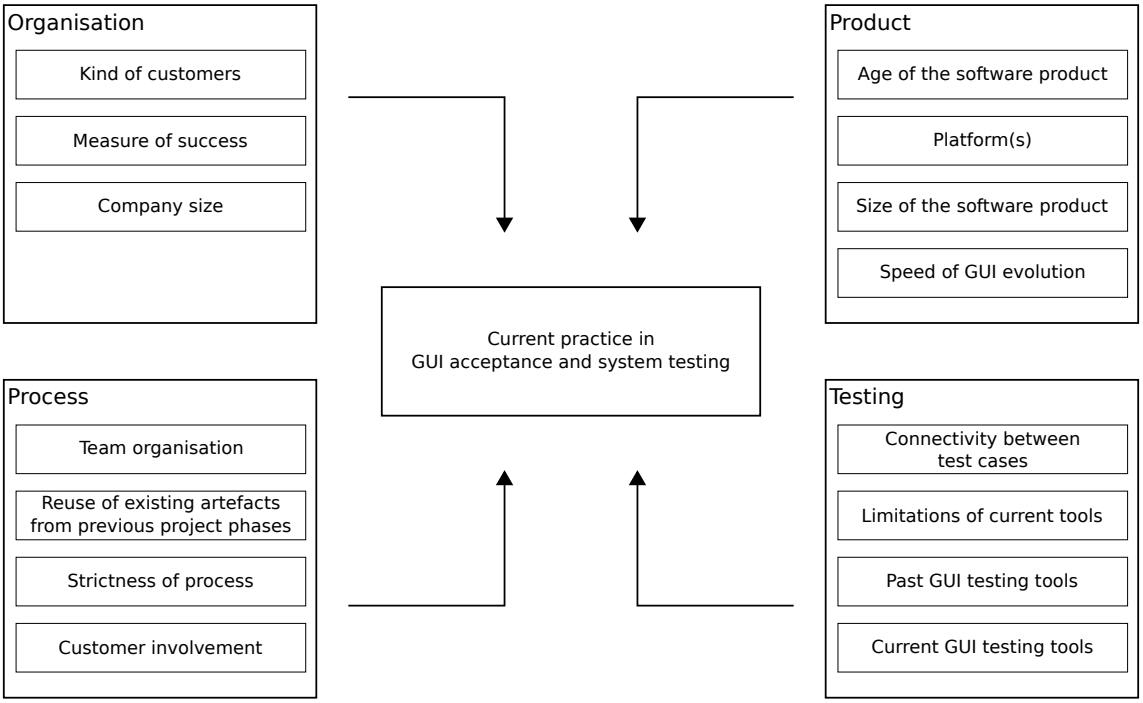

Figure 3.2.: Case study variables

Different aspects of the studied questions were covered by dividing the variables into four different concerns. These four concerns were chosen similar to the BAPO model used in software product line engineering [36, p. 16-19]. However, it seems unsuited for this area of research to sort the variables into exactly these four concerns. It is important to distinguish between the development of the actual product and the testing process. In the BAPO model, the architecture dimension would contain both product-related and testing-related variables. Therefore, the variables are instead sorted into the four dimensions of *Organisation*, *Testing*, *Product*, and *Process*. The *Organisation* dimension covers all company aspects which are not directly team-related or product-related. *Process* is equivalent to the process dimension in BAPO and contains variables both referring to testing and development processes. The two are mixed here because implemented process often overlap as well with respect to testing and software development. For instance, developers are

often testers at the same times or switch roles on a regular basis. *Product* is the dimension which covers the aspects related to the software product, in distinction to *Testing*, which covers technical variables related to testing only.
The case study questions are the set of questions which are being addressed during the data collection process and then answered separately for each company in the data analysis step later on. Note that the case study questions are not identical with the questions asked in the interviews. The case study questions are on a higher level and are answered with the help of all the questions asked during the interview. Concerning the data collection, the same set of case study questions is used in every case under study. The set of case study questions which are answered during the case study was derived from the independent variables presented before. The questions are as follows:

1. Are GUI acceptance and/or system tests used at all in the case under study?
2. What is the general understanding of the terms GUI acceptance testing and GUI system testing?
3. What kind of process is used in the project?
4. Which types of customers does the product have?
5. Are there multiple test cases which share common artefacts? That is, are there test cases which are connected?
6. Which tools are used for GUI testing?
7. Which tools have been used for GUI testing, but have been abandoned again? Why?
8. How big is the test suite for GUI acceptance and system tests?
9. How many developers and/or testers are working on the test suite?
10. Where do the scenarios for the test cases come from?
11. What are the main problems preventing the wide adoption of GUI acceptance and system testing?
12. Are there any scenarios which can not be tested using the current tools?
13. How is the test suite maintained?
14. Does the product run on different platforms?
15. Which aspects are considered by stakeholders in order to determine project success?

A number of hypotheses are used which link the case study questions with the overall research questions. The hypotheses were derived from current research in related areas and from the author's knowledge and understanding of the area. They are considered during data analysis. The hypotheses are as follows:

1. GUI acceptance and system testing is conducted in an unplanned fashion.
2. Acceptance test scenarios are only elicited by developers.
3. Customer involvement for testing purposes does not exist or is at a minimum.
4. Test suites are designed in a linear fashion. Test cases are not connected.
5. Popular automated GUI testing tools are FitNesse, FIT, and Selenium.
6. Maintenance of test suites is problematic.
7. There is a clear need for improvement of GUI acceptance and system testing practices.

8. GUI testing in general and GUI acceptance and system testing in particular does not play an important role in industry.
9. The reasons against a wide adoption of GUI acceptance and system testing are mainly missing customer interaction, missing tool support, and large overhead of test case creation and maintenance with the existing tools.

How the case study questions are linked to the hypotheses is explained in section 3.3.2.

3.2. Data collection

Based on the author's understanding of the area and the studied publications in the area of research, not enough secondary data exists to answer the first two research questions presented in chapter 1. Therefore, the first step of the thesis project is a multiple-case exploratory study in order to collect primary data from real-world examples and to ultimately answer the research questions RQ1 and RQ2. These are "How is GUI acceptance and system testing done in practice?" and "Are there best practices or methodologies in other areas of software engineering which can be directly transferred to GUI acceptance and system testing?". The units of analysis are a number of different teams within six software developing companies of varying size. For anonymity reasons they are not named here, but are instead referred to as *Company A* to *Company F*. An overview over the companies in terms of employees and kinds of customers is given in table 3.1.
The companies were selected both based on availability and on their characteristics. That

Name	Employees	Kind of customers
Company A	≈ 100	Bespoke
Company B	undisclosed	Bespoke
Company C	≈ 3000	Bespoke/Market-driven
Company D	≈ 5000	Market-driven
Company E	≈ 70	Bespoke
Company F	≈ 10	Market-driven

Table 3.1.: Companies under study

is, companies of different size, processes, and experience were contacted in order to get a variety of different contexts. Finally, the companies which expressed interest in the study were selected for the study. This corresponds to the point of intentionally selecting the units of analysis for "revelatory" reasons [53]. It can be expected that the different sizes of the companies and their special circumstances lead to a wider variety of data as if only one or multiple projects within one company or similar projects in different companies were selected. For instance, *Company C* has developed software for over 20 years and has several thousand employees. The used processes used in this company are far more established than in *Company E*, which is a small-size company using agile processes. This leads to different views on the testing practices within different company contexts. Other factors such as the kind of software and the platform are taken into consideration as well. The teams under study are located at different places in Germany, Romania, Sri Lanka, and Sweden. Some teams are spread out over more than one location.
The teams within every company were selected by the contact person at the respective company depending on suitability and free time for conducting the case study. Where possible, multiple data sources were used for triangulation reasons. The collected data is qualitative in nature. It was mainly collected in the form of interviews. According to Ghauri and Grønhaug "Unstructured interviews are advantageous in the context of discovery." [25]. Therefore, structured interviews were not used in favour of semi-structured

interviews. Semi-structured interviews helped with keeping the interviews on track and with keeping the interview time limited to one hour. The interviewees were selected by the contact person at the company together with the author. The selection was performed based on availability and suitability of the interviewees. Additional information was collected through water-cooler discussion as described in [13]. Finally, feedback on the case study reports was obtained from the interviewees and the contact person at each company. These reports summarised the interviews at the respective company, answered the case study questions, and evaluated the hypotheses presented in 3.1.
The interview questionnaire is structured in a hierarchical way. That is, questions referring to the overall picture of the company or the specific project are asked first, detailed questions about the testing practices are asked later on. The interview session is concluded with some broad questions concerning the interviewee's opinions. Even though the questions have been sorted into the four areas of *Organisation*, *Product*, *Process*, and *Testing*, they are not asked in that sorted order. This is again due to the hierarchical order of the questions. The first version of the questionnaire is depicted in appendix A. Some questions are underlined in the questionnaire guiding the author during the interviews. These are the questions which likely need more time for answering as they are broader or more detailed. This first version of the questionnaire was used for a pilot interview for companies A and B. The interview was conducted with a company proxy. That is, the interviewee is not an employee at any of the companies but has in-depth knowledge of the companies. The subject is involved in a research project with both companies in GUI test automation. Therefore, he has in-depth knowledge of the testing practices at the company. After the interview, the questionnaire was discussed and refined together with the interviewee in order to include further questions and reduced by questions which proved unnecessary, unclear, or irrelevant. The refined questionnaire was used for all interviews in Company C, D, E, and F. Apart from the pilot interview with the company proxy, eleven interviews were conducted. The interview subjects were developers and testers employed at their respective company between six months and 14 years. An overview over the interviewees and their context is given in table 3.2.
Time constraints prevented a quantitative data collection approach. Additionally, due to

Interview number	Employer	Work experience at employer	Kind of interview
1	Company C	≈ 3 years	In person
2	Company C	≈ 4 years	Telephone interview
3	Company C	≈ 3 years	Telephone interview
4	Company C	≈ 1.5 years	Telephone interview
5	Company C	≈ 14 years	In person
6	Company D	Both ≈ 4 years	Skype interview
7	Company D	≈ 2 years	Skype interview
8	Company E	≈ 5 years, 6 months, and 6 months	Group interview in person
9	Company E	≈ 3.5 and 2 years	Skype interview
10	Company F	≈ 1.5 years	In person
11	Company F	≈ 4 years	In person

Table 3.2.: Interview details

the different characteristics of the studied companies, it is not sure if a quantitative data collection would have been meaningful.
For every case studied, a case study protocol was created and maintained during the study. It consists of a short introduction to the case study, the theoretical framework, a short

description of the studied case and the corresponding company, a short outline of the data collection process and timing, and the structure of the final case study report which was created in the course of the study. The case study report was sent out to the contact persons at each company and were discussed with them. This can be seen as an additional form of triangulation as the risk of bias such as confirmation bias is reduced. *Companies A* and *B* are special cases as they were not accessed directly. Instead, the data collection was performed through an interview with a company proxy as described above. The interview furthermore served as a pilot to the interview design used in the other cases under study. Therefore, the data collected from *Companies A* and *B* can be considered a form of secondary data according to the definition used in [25]. This is due to the fact that the interviewee's knowledge about the companies' testing practices was collected for another reason, namely his own research within the companies. However, his understanding of the testing practices at the companies is deep enough to answer questions related to this research topic.
In the following, the circumstances and the data collection process are described in detail for every company.

3.2.1. Company A

Company A is a bespoke company developing safety-critical software in the area of air traffic management. The company has roughly 100 employees of which around 60 to 70 are developers. There is no distinction between developers and testers at *Company A*.
The information obtained from *Company A* was accessed through one single interview with a company proxy. The interview took approximately one hour. However, the interview covered both the practices of *Company A* and *Company B*. The interviewee had been involved with *Company A* for roughly three years in the area of testing and test automation. The interview did not target a specific project as the interviewee had a broader view of the company's practices.
The interview questions were directly derived from the case study questions in the case study protocol, which were in turn derived from the independent variables of the theoretical framework as shown in figure 3.1. The questionnaire used is identical to the questionnaire in appendix A.
The interview was recorded using a digital recorder with the author taking notes in addition to the recording. Directly after the interview, the author wrote down relevant thoughts on the interview in order to remember them correctly. The recorded interview was transcribed by the author the day after the interview. The interviewee answered the questions both with respect to *Company A* and to *Company B* where applicable. Therefore, a clear distinction is possible. Where necessary, the interviewee was asked to clarify to which company and project the answer related.
As the interview served as a pilot interview for the interviews at all other companies, the interview questions were discussed directly after the interview. The interviewee was familiar with the research topic and the thesis project. However, he was not shown the interview questionnaire prior to the interview to avoid bias. Based on the feedback, the questionnaire was refined for the following interviews.

3.2.2. Company B

Company B develops military-grade systems for their bespoke customers. As in *Company A*, the systems are safety critical. Due to confidentiality reasons, no more information concerning the systems can be disclosed. In contrast to *Company A*, *Company B* has dedicated testers who differ from the developers. The information gathering process for *Company B* was also limited to the one-hour interview with the researcher mentioned above. He had been involved with *Company B* for one year helping the company with the automation of manual system tests.

3.2.3. Company C

Company C is a global company developing ERP software. Their software targets mid-size to large-size global companies and is available in multiple languages. The product is developed at multiple sites in different countries. The company can be seen both as bespoke and market-driven as their product is developed for a general market, but consultancy is offered for single customers. The case study at this company was actually divided into multiple studies. Two different teams working on different parts of the company's products and therefore with a different environment were interviewed. Within the second team, an approach on how to do automated GUI testing is currently being developed apart from the normal ongoing development. In each team, multiple team members were interviewed for triangulation reasons. Facts that could not be captured during the interviews, were not understood properly by the interviewer, or were simply forgotten during the interview were clarified afterwards using email. A non- disclosure agreement was signed with the contact person at the company.
There were a total of five interviews at *Company C* with interviewees from two different teams. The interviewees were not shown the questions prior to the interview. Before every interview, it was clarified that the interview would be completely anonymous and that no sensitive data or answers would be published or discussed in detail with colleagues. Each interview was recorded using a digital recorder after asking for permission to do so. Furthermore, notes were taken by the interviewer. Out of the five interviews, two were conducted in person and three over telephone conference. This was due to availability and geographical reasons. The interviews were transcribed as shortly as possible after the interview by the interviewer, not later than the day after the interview.

3.2.4. Company D

Company D is offering, among other things, web hosting services. Therefore, most software development is in the form of web application development. The company is market-driven. The case study at this company took place in the quality assurance department of the company's web hosting division. The contact person at the company works in another area, but decided that the quality assurance department would suit the research better and benefit from it more. Due to geographical reasons, on-site visits were not possible. The study was therefore conducted over telephone conferences. The data was collected in two interviews, whereas follow-up questions were answered using email.
Two interviews were conducted at *Company D* for triangulation reasons. The first interview was conducted with a team leader and a senior quality assurance engineer. A second interview was conducted with another quality assurance engineer. The interviews were conducted using Skype [4], including its video functionality. Both interviews were recorded using a software tool after asking for permission to do so. The interviews were transcribed the day after.

3.2.5. Company E

Company E develops software and does consultancy services related to software development. The development is done using agile processes. The company does not have an own product but rather develops entire products for bespoke customers. Their focus is on web applications. At *Company E*, a group interview with three developers was conducted. Additionally, an interview was conducted with two more developers. The interviewees had been employed at the company between six months and five years.
The questions were asked with a team focus and not focused on a single software product. This was due to the fact that the employees had been involved in several projects with only a short duration of typically one year or less for each project. As similar techniques with

regard to testing had been used in all projects, most questions were still suitable. During the group interview, notes were taken and the answers to the questions were directly written out in detail after the focus group. Things which were unclear were validated by email with the group members afterwards. The second interview was conducted using Skype including its video functionality. The interview was recorded using a software tool after asking for permission to do so. The interview was transcribed closely after it was conducted.

3.2.6. Company F

Company F is a market-driven start-up company developing software for the security sector. The development is done using agile processes. The product runs on different platforms. At *Company F*, two interviews with two different developers were conducted in person. The experience of the interview subjects ranged between 1.5 and four years of experience in the company. The interviewees were chosen by the contact person at the company based on availability and suitability.
Both interviews were recorded using a digital recorder after asking for permission to do so. Furthermore, notes were taken by the interviewer. The interviews were transcribed directly after. Unclear things were clarified through email.

3.3. Data analysis

The data analysis is conducted mostly on the transcribed interviews as this is the primary data source in the performed multiple-case study. As described in [63], the aim of the analysis is to answer the case study questions stated in the case study protocol. These are the same questions as presented in section 3.1. From the answers to these questions, every hypothesis is checked for validity. As the interview questions were derived from the theoretical framework, the answers can be directly connected to areas in the theoretical framework. This ensures a clear chain of evidence in the analysis.
The analysis was done manually without special tool support apart from word processing software. In the following, the data analysis is described. Firstly, the coding of the transcribed data is presented. Secondly, the approach of answering the case study questions and judging the case study hypotheses is described. It is explained how the interview answers are linked to the case study questions and hypotheses through the theoretical framework and how data triangulation is assured.

3.3.1. Coding

The interview transcriptions were coded using simple codes describing the area to which the answers referred. The codes were not determined a priori, but during the coding process. However, it was tried to keep the codes linked to the independent variables and areas in the theoretical framework depicted in figure 3.1. After all the transcriptions were processed, the codes were reviewed. Two codes were removed because they were semantically identical or similar to other codes. The interview parts coded with these two codes were reviewed accordingly. The following codes emerged after the coding process. The two codes removed after the reviewing process are marked grey.

1. Company size
2. Kinds of customers
3. Target sector of product
4. Team organisation
5. Product attributes
6. Test suite size
7. Test automation
8. Requirements representation

9. Role and experience of interviewee
10. Company experience and background
11. Way of working
12. Process influences
13. Manual testing
14. Customer involvement
15. Definition and understanding issues
16. Testing levels
17. Problems with testing
18. Process acceptance
19. Scenario elicitation
20. Test representation
21. Traceability
22. Test connections
23. Maintenance
24. Influences from external roles
25. Priorities within the team or company

Code number 10 was removed because it was of no direct use for the answering of the case study questions. Parts of the transcriptions could directly be coded using other codes like number 2,3,5 or 9. Concerning code number 24, it was often not clear if a passage should be coded using code 14 or 24. Instead, it was decided to remove code 24 and use code 14 for issues concerning customer contacts. For other influences, code number 12 was used instead.
The codes were assigned to every sentence in the transcribed answers. Hereby, a sentence could have different codes attached to it and a code could be assigned to different sentences. After the coding, the appearances of the codes in the transcriptions were counted. This was done in order to get an overview over the code usage and for reviewing the codes. Throughout the coding, a document listing the codes with a short memo or description of the respective codes was used.
After reviewing the codes and modifying the coded transcriptions accordingly, the coded sentences were sorted by code. This facilitated the following process of answering the case study questions.

3.3.2. Logical linking

In the following, the chain of evidence between the interview answers and the theoretical framework is discussed. As discussed in section 3.1, the theoretical framework is connected in itself. That is, the independent variables are linked to the case study questions which are in turn linked to the case study hypotheses. From the case study questions, the interview questionnaire was derived. It primarily aimed at answering the case study questions. Additionally, some questions were asked concerning general information about the interviewee and the company. For instance, the size of the company is not directly related to the case study questions, but enables the author and finally the reader of this thesis to understand the circumstances within the units of analysis.
The codes listed previously were used directly to answer the case study questions. For this, a matrix was created linking every case study question to a number of relevant codes. Every question was associated with a number of codes based on the author's understanding of the area. Then, the sentences in the interview transcriptions coded with these codes were used to answer the case study questions. Other parts of the interviews not coded in one of the relevant codes were not considered for the question's answer. The table linking case study questions to codes is depicted in table 3.3. Please refer to the previous subsection for the code numbers and to section 3.1 for the case study question numbers.
As it can be seen, some codes have not been used at all. Apart from codes 10 and 24 which have been removed, these are the codes related to contextual information which is not directly relevant to the case study questions. For instance, questions regarding the company size and the sectors for which the company offers products have been asked, but are not

	Code															
Question		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2				X											
	4			X						X						
	5														X	
	6	X							X							
	7						X									
	11			X												
	12			X				X								X
	13	X														
	14	X		X												
	15		X													
	16	X														
	17							X				X	X			
	19										X					
	20					X	X									
	22					X										
	23							X						X		
	25			X								X				X

Table 3.3.: Codes relevant for case study questions

used for answering any case study questions. Even though these codes are not relevant for the analysis of a single case, they can enable the author to explain differences between the single cases under study. During the design of the PASTA framework, sentences coded using these codes were considered again.
Starting from this table, the case study questions could be answered using interview transcriptions sorted by code. Where possible, it was tried to verify statements given by one interviewee using confirmatory statements from other interviewees within the same unit of analysis.
Given the answers to the case study questions, the case study hypotheses could be evaluated. This was again done by linking the case study questions to the hypotheses in a similar fashion as done with linking codes to questions. The relevant case study questions for each hypothesis are shown in table 3.4.

	Hypothesis															
Question		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	X	X	X			X		X							
	2	X	X	X						X	X					
	3				X											
	4					X	X				X	X		X	X	
	5						X	X								
	6											X		X	X	
	7			X			X	X			X	X	X	X	X	
	8	X	X	X					X			X				X
	9	X					X	X	X		X	X	X		X	

Table 3.4.: Case study questions relevant for case study hypotheses

This overall linking from single interviews to the final hypotheses is outlined in diagram 3.3.2. The diagram follows the style used in [53].

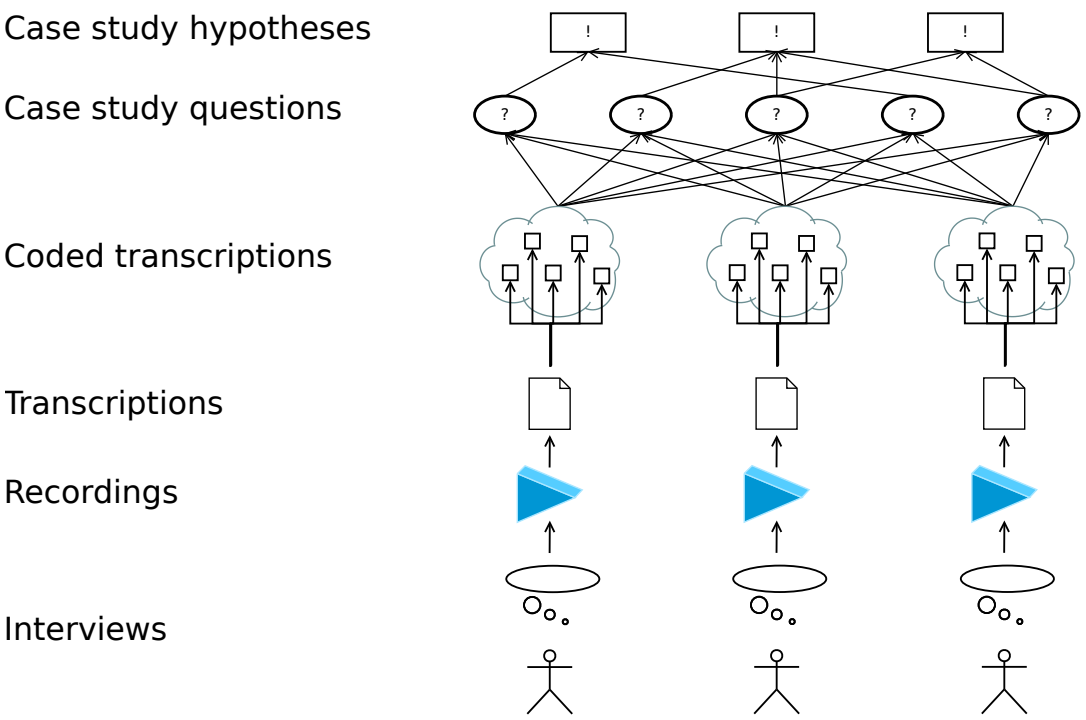

Figure 3.3.: Information flow

4. Data analysis and framework design

This chapter discusses the outcome of the data analysis performed within the multiple-case study described in the previous chapter. Furthermore, it contains the design of the PASTA framework derived from the information gained in the case study. The following chapter is organised as follows. Firstly, the case study questions are answered. This is done separately for each company. Secondly, the case study hypotheses are evaluated based on the answers to the case study questions. The hypotheses are evaluated generally, not focused on single companies. Thirdly, the case study findings are shortly summarised. Fourthly, the design of the PASTA framework will be presented. For each part of the framework, it will be explained how it was derived from the case study and which rationales are behind it.

4.1. Analysis of case study questions

In the following, the case study questions stated in the theoretical framework in chapter 3 will be evaluated. They are answered separately for each company as they depend on the company context.

4.1.1. Case study question 1

Are GUI acceptance and/or system tests used at all in the case under study?

Company A
Company A runs both system and acceptance tests. The tests are conducted on GUI level. They are currently working on automating the tests, but so far it is manual testing only. The acceptance tests are run by the customers, and are conducted both as factory and site acceptance tests:

> "They have factory acceptance tests, and then they have site acceptance tests. Factory tests are with the reference systems, the site acceptance tests are on location."

The customers are also supposed to bring their own test scenarios to the acceptance tests which are then tested manually.

Company B
Company B runs both system and acceptance tests. The tests are conducted on GUI level. They have partly automated their system tests using the visual GUI testing tool Sikuli [17]. However, most of the testing is still done manually. The acceptance tests are run by the customers similarly to Company A:

"The acceptance tests are run completely, more or less, by customers in (Company B) as I've understood it or they're part of it at least."

Company C
In the first team, system testing on GUI level is conducted. The tests are automated using NetBeans JellyTools [7]. There are around 30 test cases on system level, covering most of the important functionality. Some of the functionality not covered by the tests is being tested manually. However, this is no formalised process. There is no acceptance testing from the customer side. Informally, there are some demos and users using it live, but this is again no official process. Usually, there is some sort of feedback loop with users and/or the customers, but no regular meetings.

"We release the product and then wait for feedback, but we actually don't do user acceptance tests."

In the second team, manual system testing is done with around 150 test cases. Automation of these tests is wanted for the future. An approach for automated GUI testing on system level is being developed within the team at the moment. With this new approach the main process flows are supposed to be tested.

Company D
Acceptance and system testing is performed at Company D according to the interviewees:

"If you think of test levels like unit, integration, system, and acceptance, I would say that we definitely cover the last two."

However, one of the interviewees stated that they sometimes focus more on system testing. Within some of their projects, they work in a process close to a waterfall process. In these projects, the team conducts manual acceptance and system tests. In other projects, the work is done following agile principles. Then, they also have automated tests. These are on system level using Selenium WebDriver [3].
Customers are not in direct contact with the quality assurance team. According to the interviewees, they are sometimes involved in beta testing the web applications. This can be seen as manual acceptance testing. However, there is no customer involvement when it comes to acceptance testing done by the quality assurance team itself.

Company E
System testing at Company E is done in the form of automated Selenium WebDriver tests. In general, it is tried to have a pyramid structure where the biggest part of testing is done on unit testing level and only a few system tests exist. However, they are an established practice at Company E. It was said that developers write their own acceptance tests during a sprint using Selenium WebDriver. As the customer is not involved in this practice, it could also be described as a form of automated system testing.
Acceptance tests are done manually by the customer after each sprint. During these meetings the customer clicks through the application and checks if the required user stories were completed. Apart from the abstract user stories, the test cases are not documented. The degree of customer involvement depends on the project. Also, the focus of the customers changes a lot depending on the project. Some customers have been heavily involved in the testing, other customer were only concerned with the look and feel of the system's GUI. The team stated that they would like to do acceptance-test-driven development with more customer involvement in the future in order to increase their testing quality.

Company F
According to both interviewees, acceptance testing of any form is not used in the company.

However, manual system testing through the GUI is done. The testing is done ad-hoc by trying to test the new functionality in the system and to break it.

" [..] what we usually do right now is just take all the affected parts of the system and then try to break them."

Even though the interviewees did not name it, this practice can be seen as exploratory system testing [59].
Additionally, there has been beta testing by customers in the past. This could be seen as a form of manual acceptance testing. Whether beta tests are still conducted was not clear from the interview data.

4.1.2. Case study question 2

What is the general understanding of the terms GUI acceptance testing and GUI system testing?

Company A
The interviewee stated that a system test aims at "verifying system conformance to requirements", whilst an acceptance test aims at "validating conformance to requirements". The difference between the two is, according to the interviewee, that a system test makes sure that the system works, whilst an acceptance tests makes sure that the system works in the way the customer wanted it to work.

Company B
As the same person answered the questions for Company B as for Company A, the answer to this question is the same as for Company A.

Company C
For system testing, the interviewees agreed that it concerns testing the whole system, including all components and interconnections between them. On the other hand, there were different opinions concerning the timing of system testing. Whilst most interviewees did not specify when system testing is happening in their definition, one interviewee mentioned that it should be performed at the end of the project. One interviewee noted that a system test should contain the "Typical workflow for a user.", whilst another one said "[..] it kind of ensures what the customer required.".
For acceptance testing, there was no consistent definition across interviewees. The interviewees said that usually the tests should be conducted by a different party, for instance the customer or the user. One interviewee said that it should check that requirements are fulfilled.

Company D
One of the interviewees stated that system testing means doing end-to-end testing.
The interviewees mentioned additionally to this that system tests are testing functionalities which are affected by changes. In contrast to this, acceptance testing would only focus on new features. In general, they brought up that acceptance testing is focused on positive scenarios and "proving that it's working".
Another interviewee stated that system testing focuses on testing single systems if the overall system is composed of different subsystems:

"[..] when you have (a) more integrated system sometimes you are not able to test them at the same time. So you have to test them one by one."

Company E
For system testing, it was said that the tests should test the whole system through the user interface. This would include all layers of the system.
Regarding acceptance testing, the interviewees were not sure. One idea was to check if user stories are finished. Another proposed definition was "a test to check if features have been implemented.". One interviewee also brought up the question whether acceptance testing should be done on GUI level or on the level directly under the GUI.

Company F
The understanding of the interviewees regarding acceptance and system testing were close to the definitions. One interview stated:

"System testing would be the user's view of the complete system, I would say. That you test the integrated system."

For acceptance testing, it was stated that the user or the customer needs to be involved in order to verify that the requirements have been fulfilled.

4.1.3. Case study question 3

What kind of process is used in the project?

Company A
Company A uses a number of different processes depending on the project. They have projects which work in a plan-driven manner, agile projects, and they also try out new things like Kanban:

"In Company A they have a toolbox of processes used for different projects. So some projects are more plan-driven than others. They have recently introduced Kanban, they used Scrum. So they tried different things and work with different things."

In Company A, the testing is conducted by developers.

Company B
Company B uses a plan-driven process close to a waterfall process. This is mainly due to the company's history and because of their work in the military sector. Within every iteration, there are currently around 8 to 12 man-weeks being spent on manual testing only. This huge effort is mainly justified through the fact that Company B develops safety-critical software. The testing is performed by dedicated testers at Company B.

Company C
Within Company C, an adapted version of Scrum is used. This process is used in all interviewed teams within the Company. They have monthly iterations and there is no difference between testers and developers.
Within the first interviewed team, releases happen approximately every week. JIRA is used for managing requirements, priorities and bugs. Jenkins on different platforms is used as a continuous integration server including automated testing. Demos, user meetings, and retrospective meetings are not always happening in contrast to start-up meetings before every iteration. These happen before every sprint without exceptions. The customers are developers within the same company mostly, so there is a close connection between the developers and their actual users or customers. Testing is done in a test-first way where the test has to be written before new functionality is introduced [10].
Within the second team, there are around 20 developers. There is also no difference

between developers and testers, but there are sub teams for different areas. JIRA is used as well for issue tracking and bug reporting by the users.
Once a month, binary patches for the framework are released. These are tested twice a month using around 150 manual tests. There is again no direct customer involvement, but the team gets feedback from users.
The GUI automation approach which is developed in the second team is also aimed at fitting into the agile process used within Company C. However, the team itself uses an adapted process which fits their needs. For instance, iteration times differ depending on the project state. Within the team, there are four developers working on the GUI automation approach. However, they sometimes get help from people in other areas. There is no direct customer involvement and often things are not directly being paid for by customers. Sometimes they involve other people in the role to act as a customer.

Company D
The quality assurance team is following different processes depending on the project they are doing quality assurance for. They are not only focusing on testing, but rather quality assurance in a broader sense. That is, they also define general processes in order to improve a product's quality:

> "What we are doing is not only testing, but QA meaning defining processes for the improvement of the quality of a product."

Some projects have a process following the waterfall process. In these projects, the quality assurance team is reviewing requirements, giving recommendations for the developers' testing practices and doing the testing on system level. That is, system and acceptance testing. In these projects they test only in a manual way. This is due to the fact that the testing is not done more than once. Therefore, the overhead for creating automatic test suites is seen as too high. Being involved in this kind of projects also means that the quality assurance team is not involved in development, so there is a distinction between developers and testers. The periods for testing and quality assurance in general depend on the size of the project:

> "Between one day [..] until two months of QA. That was the record."

In agile projects, there is no distinction between testing and development. So there, the quality assurance team is included in the development process. Similarly, there is also not a single test phase but the involvement goes on for the whole project period. The team usually follows Scrum within the agile projects.

Company E
The company uses Scrum as a process. They use two-week sprints with no distinction between testers and developers. The Scrum Master is also a developer within the team. The team is supposed to work in a test-driven manner. They have a pyramid-like automated test structure. That is, unit tests are written using JUnit with the aim of reaching a high code coverage. Then, they have a small number of integration tests on top of that. Finally, they have a couple of system tests written in Selenium. These tests are run before every build on a Jenkins continuous integration server. If a build breaks, the developer who broke it is responsible for fixing it again.
The customers bring user stories for each iteration and perform manual acceptance tests in meetings after each sprint.
The interviewees stated that they are flexible in changing things in their process as long as the changes are not too big. Employees can propose changes to the process.

Company F
The team uses agile development processes. They used to follow an approach similar to Scrum with four week sprints. Before every sprint there was a planning phase where user stories were prioritised and divided into tasks. The user stories were derived from customer requests. However, they did not follow Scrum strictly but adapted the sprints to their current situation. In general, it was stated that the company is very flexible with changing things in the process.
Right now, there is only improvised exploratory testing after each sprint. Hereby the developers do not test their own code.
One interviewee stated that in the last years they mainly aimed at releasing new features very regularly. Therefore, there was not enough time to establish processes and testing:

"[..] for the last couple of years there has been a major race just to get the product in a usable state and delivered, and include most of the customer requirements. So just coding and pushing out releases have been the focus. To deliver. So now, this year and coming years, we have sort of slowed down. Now it's almost feature-complete. Now we can take time to improve our processes and improve our testing. So we are in the phase of introducing a lot more processes."

According to one interviewee and to the contact person at Company F, the company is right now shifting to a new process. They try to work more feature-oriented.

4.1.4. Case study question 4

Which types of customers does the product have?

Company A
Company A delivers to a national military service provider and a national civil aviation authority. Their end customers are individual airports both in their own country and international.

Company B
Company B has customers in the military sector. Due to confidentiality reasons, no more details can be disclosed here.

Company C
The product developed by the first interviewed team is being used by developers within Company C. Furthermore, the tool is sometimes bought by customers outside of the company for development. However, they are usually consultants working for Company C as well. The main part of the customers is the internal developers, though.
The same situation is found in the second team. The group develops the framework used by developers working on the components which make up Company C's final product. Here, no external customers were mentioned by the interviewees.

Company D
Company D is offering web hosting products to its customers. The range is big depending on the exact product. It could be customers with a deep technical knowledge administrating servers to customers with no technical knowledge building their own web site through simple site builders.
There are no formal customers for which the products are developed, but rather a market to which the projects deliver.

"We have many many clients, so it's not like in other companies where you work with a real client and you know exactly what he wants. There are marketing researches that

identify the needs and then product management creates the project ideas for the new features that we should offer to stay competitive"

Company E
Company E does software development and consulting for and with bespoke customers in different areas. The area depends on the customer, but the team stated that they have been active in various different sectors in the past. The projects typically run one year or less and are placed in the area of web development.

Company F
The company's customers are based in the security sector. The company's product is facilitating the customers' work when dealing with their end clients. Now, the company is also planning to transition into an additional sector where their product could be used without substantial changes.

4.1.5. Case study question 5

Are there multiple test cases which share common artefacts?

Company A
In Company A, the tests all test different parts of the system's functionality. Therefore, they do not share any artefacts. This is mostly because the functionality of the system is placed in very different components of the system. The tests are however grouped depending on which features the tests aim to test.

Company B
Company B has introduced an overall structure which depicts the relationships between test cases. Within this graphical representation, the test scenarios are split into smaller units. These units might then be used by different test scenarios. They also use different levels of their graphical representation.

"Use cases are tied together to test chains which in turn then include test paths. And each test path is a test case. They have an overall visual representation of how the use cases are tied together."

Company C
In the first team, basic reuse principles are applied for automated testing. That means, common parts of test cases are extracted into parent classes which are then inherited by test classes using that common functionality. As an example, they mentioned the scenario of opening a project, which occurs at the beginning of most system test cases. One interviewee stated:

"Mostly we try to reuse what is there. But if the reusing process is quite cumbersome, we might copy paste it. And if the reusing stuff doesn't 100% fit then we have to copy paste. And we do copy paste."

Regarding test cases with overlapping sub-scenarios, it was said that usually the aim is to model the test cases independently, so that program flows can be tested in isolation:

"If there are relations we try model from scratch and derive a separate test case for that so that the whole process can be tested in one flow and that flows can be tested isolated."

Within the second team, the testing is done manually. Within the manual test suites, no shared artefacts as such exist. However, the test cases are related in a way that they are grouped by an area and by a topic.
The ten test cases which were so far implemented in the GUI automation project are interconnected. They do not share any artefacts, but rather have a natural order in which they have to be executed.

Company D
In the automated test suites, test cases share common parent classes and utility classes. That is, code is being reused. Apart from this, there is no sharing of common artefacts. Test cases are grouped by test suite and it might be that one test case forms the precondition to following test cases. That is, there is a logical order in the test case execution. This logical ordering is done both in the manual and in the automatic tests.

Company E
The team uses code abstraction layers for their automated testing, but they do not directly connect test cases through common artefacts apart from this. They state that they do their testing "Like software engineering in testing".

Company F
Tests at Company F are not documented. Therefore, there is no sharing of artefacts of any kind.

4.1.6. Case study question 6

Which tools are used for GUI testing?

Company A
Company A is in the process of transitioning into using Sikuli [17] with the help of the interviewed researcher. However, it was so far not used in a running project. Apart from Sikuli, the company does manual testing only without any mentioned tool support.

Company B
Sikuli [17] is being used in Company B for automated testing. The tool was introduced roughly one year ago and so far only a small part of the company's test suite has been implemented in Sikuli. For manual testing, no tool support was brought up by the interviewee.

Company C
Within the first team, the NetBeans IDE and its Jelly functionality are used to write user interface tests. Jelly, or Jemmy on which it is based, is able to manipulate components using Java Swing. The tests are written in a JUnit-like fashion and can be executed automatically by the team's continuous integration servers.
In the second team there is no GUI testing tool. However, test cases are collected and handled using Microsoft Excel and JIRA.
Finally, within the developed GUI testing approach the WindowsAutomation framework is used. Test cases are implemented using Visual Studio 2010 premium edition and are executed using MSTest. The framework is using a sort of capture and replay mechanism identifying user interface controls. In addition to being able to execute the tests, they can also be distributed over several machines and executed on all of them. Logs, screen shots, or even movies of the execution process can be collected using MSTest.

Company D
For system testing, the team is using Selenium WebDriver.
For managing the test cases, the tool TestLink [6] is used. This is no GUI testing tool as such, but is used in context with the system tests. Therefore, it is mentioned here as well.

Company E
For GUI testing, the team uses Selenium WebDriver. The tests are written in Java.

Company F
Right now Company F does only manual GUI testing. GUI testing tools are not in use.

4.1.7. Case study question 7

Which tools have been used for GUI testing, but have been abandoned again? Why?

Company A
In Company A, an own tool was developed for automated GUI testing. It was abandoned eventually because it was too costly. Maintenance was difficult with this tool:

"In Company A they developed their own tool once upon a time which aimed to do this. It was based, as I understood it, on unit tests. So it was very big and complex and horribly hard to maintain. So they just abandoned it."

Company B
The interviewee was not sure if Company B ever used any other tools for GUI testing apart from manual testing and Sikuli. He mentioned that they evaluated a capture and replay tool at some point:

"They talked about using some capture and replay, but that was just some minor evaluation."

Company C
The first team used NetBeans functionality throughout the interviewees' participation in the project. One interviewee remembered a discussion were it was said that Jelly was chosen because the previous tool was based on NetBeans functionality as well.
In the second team, one interviewee mentioned that RationalRobot was used more than ten years ago. It was finally dropped because it was not reliable enough and maintenance became too complicated:

"[..] what we were doing was actually using TAB to move between the fields to find the right one. And, as you understand, very soon it gets messy and then you don't rely on them. So it was build up a little bit and then it was too complicated and tests died. Maintenance problems."

Company D
The team used a tool called TestComplete some time ago for GUI testing. However, they discarded it eventually because it was outdated. Every year, a new license had to be bought because new browser versions were not supported any more:

"Well, for TestComplete, it was outdated. We had to buy every year the license for a new version of TestComplete because it didn't support the new version of Internet Explorer for example."

Company E
One of the interviewees used the record and replay functionality of Selenium in one of the projects he was involved in. However, tests were breaking all the times when things in the GUI changed. Therefore, it was abandoned again.

Company F
According to both interviewees, there have been some tool evaluations at Company F. However, both employees were not involved in these evaluations and could therefore not provide any further information.

4.1.8. Case study question 8

How big is the test suite for GUI acceptance and system tests?

Company A
Company A has approximately 65 manual system test cases for their system. The test cases vary greatly in execution time, from approximately 30 minutes up to five hours or more.

Company B
The system test suite for Company B's project is very large compared to the other cases under study. They have 40 test suites which are build from around 4000 use cases. Every iteration, they do manual system testing for about 8 to 12 man weeks. Part of this test suite has been automated in the last year.

Company C
In the first team, the test suite for system tests consists of roughly 30 automated test cases. They vary in size and execution time between short ones, like opening a window, and longer ones which can take several minutes. According to one interviewee, most of the important functionality is covered, but there are more things that could be covered:

"I think we have quite a few more tests that we could cover. I mean coverage is not what we would like to have, but we have most of the important things covered."

The second team has around 150 manual test cases for their product. These are run twice a month at the current state.
Within the new GUI automation project, ten process flows are currently covered. This means there are ten test cases. However, they have to be executed in a logical order and can not run without the preceding test cases having completed successfully. Within the ten test cases, around 70 validations are done. The whole test suite currently takes about 30 minutes to execute.

Company D
The test suite size depends on the size of the project the interviewees' team is doing quality assurance for. One interviewee stated that the sizes range from about 50 to 100 test cases in small projects up to 5000 test cases in big projects. In large-size agile projects, prioritisation is usually necessary as not all the test cases can be covered with automated tests during the quality assurance period. Then, the test suite consist partly of automated tests and partly of manual tests.
One interviewee involved in manual testing stated that a medium-sized project with approximately 20 man days of testing would have around 700 test cases. However, the individual test cases are kept short and simple.

Company E
The test suite size depends on the project. However, a typical size for automated system tests is between 10 to 15 tests written for Selenium WebDriver. The acceptance testing is done informally and is not documented.

Company F
There is at the moment no formalised test suite for GUI system and acceptance tests at Company F.

4.1.9. Case study question 9

How many developers and/or testers are working on the test suite?

Company A
One developer is dedicated to do manual system testing at Company A. This developer changes, so it is not always the same person dedicated to manual system testing. During factory acceptance tests, more employees and the customer are involved in the testing.

Company B
Company B assigns two to three people on running through the manual tests. For the factory acceptance tests, the interviewee was not sure how the process is at Company B.

Company C
There are six to seven employees working on the first team's product. They are doing both development and writing automated tests.
In the second team, there are two developers responsible for the test repository. However, the manual testing is performed not only by these two developers.
The question does not apply to the GUI automation approach, as it is not established, yet.

Company D
The interviewees' team consists of ten people of which one is the team manager. The whole team is doing testing.

Company E
The team size depends on the project. However, there are between 10 to 15 developers at the visited company site. As the team follows Scrum, all developers also write and execute automated tests.

Company F
According to one of the interviewees, there are five developers. All developers do exploratory manual testing without documenting or discussing the test cases.

4.1.10. Case study question 10

Where do the scenarios for the test cases come from?

Company A
The test scenarios at Company A come from multiple sources. Firstly, the customers bring their own scenarios to the acceptance tests. Secondly, the developers develop new test scenarios from their own domain knowledge. Finally, there are many tests which have existed for a long time. Most likely, they also have their origin in customer or developer scenarios.
The interviewee stated that it could also be that test scenarios have been derived from bugs in the past.

Company B
The system in Company B is over 20 years old. Therefore, most scenarios have existed for a long time and new ones are usually not developed. It can therefore be stated that the scenarios come from legacy.

Company C
The tests in all of the company are connected to so-called process models. By writing documentation, test cases are automatically generated. Therefore, the test scenarios are elicited once the functionality is written. This means, of course, that the scenarios come either directly from requirements or from each developer's domain knowledge. The first team also generates test case stubs for their automated test suite.
In two interviews it was mentioned that sometimes a new test case can be created based on a reported bug. However, this is not always happening.

Company D
The engineers within the quality assurance team create the scenarios from several documents and their own understanding. They have a document describing the high-level requirements, the new features within a product, and other relevant information regarding the product. Another document specifies the look and feel of the product, that is the graphical layout. Finally, the product's features are described in another document. Starting from these three documents, the engineers define their own test cases, which are then reviewed by the team leader. According to one interviewee, the requirements are also reviewed by the quality assurance team within the waterfall projects in order to make sure they are clear enough:

"We do check the requirements and come up with issues that are not clear enough. We are involved in the planning, let's say, in the position of test manager."

One interviewee stated that in large projects they also discuss the test cases on an abstract level with project and product managers. This leads to extra input from their side. Even in smaller projects, the quality assurance team is in direct contact with product management and can clarify things if necessary.

"And also you are in direct communication with product management and if there is something that you don't understand you just call him and try to clarify what that functionality should do."

The interviewees stated that the functionality they test is usually not a new idea, but already exist in other products or previous versions. Therefore, the test cases can be based on experience and do not have to be elicited entirely new.

Company E
Scenarios in the form of user stories for the test cases are provided by the customer. However, these scenarios are vague. The developers then extend the user stories to full scenarios which can be automated by them.

Company F
Every developer comes up with his own test scenarios based on his understanding of the system. Additionally, one interviewee stated that they try to test the code of other developers instead of their own code in order to not overlook things.

"So we try to test each others code at least. So you don't sit and test your own."

4.1.11. Case study question 11

What are the main problems preventing the wide adoption of GUI acceptance and system testing?

Company A
Concerning automated tests, Company A used a self-developed tool in the past. However, it has proven to be too costly. The fact that the company developed its own tool points at missing support from existing tools in this area. However, the interviewee brought up that with Sikuli it can also be frustrating to work, as it is not very mature, yet.

"However, Sikuli is not really a mature product, to say the least. It's not even a beta yet. So it has a lot of bugs which has been found quite frustrating for new users. So you sort of have to stick with it for a bit."

Concerning improvements in general, the interviewee stated:

> "Obviously there is a lot of room for improvement. The thing is that you have to consider cost and time constraints. So what you have to consider is the amount of quality you can reach for a certain amount of resources."

So, apart from missing tool support, the costs of introducing a test suite and conducting the tests is another factor preventing the adoption of GUI acceptance and system testing. Manual GUI acceptance and system testing is an established practice at Company A and specific problems with regard to the manual tests were not stated. Therefore, this case study question does not apply to the manual tests at Company A.

Company B
Using system and acceptance tests on GUI level is established in Company B. Therefore, the question can not directly be answered for Company B. Regarding automated GUI acceptance and system testing, it has to be noted that the manual test suite is already old and most scenarios come from legacy. However, automation has only been introduced recently. As the interviewee stated, Company B only evaluated a capture and replay testing tool according to his knowledge. It can therefore be assumed that the tools were considered not mature enough during that evaluation and manual testing seemed to be more cost efficient. Otherwise, automated GUI testing tools would have been adopted earlier most probably.

Company C
In the first team, three main factors were identified during the interviews which prevent more or better GUI acceptance and system testing. Firstly, problems with the Jelly framework prevent the implementation of some test scenarios. For instance, interaction with diagrams proved cumbersome. Furthermore, it was stated that only little documentation for Jelly is available and therefore a lot of experimenting is necessary to work with it. This will lead to less time available on actually writing new tests, which leads to a smaller test coverage. Manual tests as a potential alternative were not brought up by any of the interviewees. Hence, it seems that they favour a small automated GUI acceptance and system test suite over having any form of manual testing.
The second point mentioned by the team are problems with the current way of working. Firstly, test-driven development was stated to be a practice hard to get going and to see benefits from it. As a possible improvement, behaviour-driven development was mentioned by one of the interviewees. Secondly, the general attitude towards the agile process was mentioned to be not good enough, yet. It was said that other parts of the company, like management, would not appreciate the focus on quality highly enough. Another person noted that the focus within the team was also not strongly enough on testing, yet:

> "I think still we need to improve our attitude towards testing. Still we are more [..] developers and if we have extra time we do testing. There is not enough focus on writing tests."

This point is relevant for both automated and manual testing.
Thirdly, the missing interaction with users and customers was mentioned as one of the factors preventing a wider adoption of GUI acceptance testing. Of course, acceptance testing itself can only be done if there is some form of customer interaction. In the context of writing tests in a behaviour-driven way, the missing customer interaction was also mentioned:

> "How can you verify it's working if you don't know how it's supposed to work?"

The team clearly stated the need for a closer interaction with customers or users in the form of regular, formalised meetings including an acceptance process.
In the second team, there are so far no problems with automated testing tools, as the testing is fully manual. However, it was stated that the regular testing twice a month tires the people working on it. Therefore, a strong need for test automation regarding GUI acceptance and system tests is expressed from multiple sides. Furthermore, some of the test cases take a long time to be run manually and have to be sped up:

"Because we doing it twice a month, people get tired of the test cases. Some of them are too big. It takes too long time to run through the test cases. So we're working to speed up the test cases, to easier descriptions."

A number of issues in automated GUI testing were brought up regarding the GUI automation project. Interestingly, some things related to the process were stated, which are similar to the ones mentioned in the first team. The interviewee stated that there are some problems with the company having moved from a classical, plan-driven approach to agile principles. This was mainly due to the fact that still not everything was adopted and established properly. So even though there is supposed to be an agile way of working, a lot of things from the previous processes can still be seen. This could explain the *attitude towards testing* mentioned above in the context of the first team. Again, this problem should also be taken into account with regard to manual testing. This is due to the fact that it concerns the developers' mindset and not whether the testing is automated or not.
Also, customer interaction was mentioned as a problem. As not every project is paid for by a customer, it is hard to get them to participate in these projects. Therefore, the team sometimes tries to replace them with other people acting as customers:

"Because [..] not everything that we're doing is directly paid (for) by the customer. So that's why we don't have it in every case. We try to have other people which sort of act as a customer in these cases. But not every time I would say."

Finally, a lot of issues regarding tool support were mentioned. Problems with the WindowsAutomation framework prevent to automate tests properly. Some controls in the graphical user interface need to be accessed using indices or screen coordinates because WindowsAutomation API is missing for them. This can often not be controlled by the company, as it happens both in the own code, but also in third-party components. Access through indices and screen coordinates is fragile and can lead to tests breaking even with small code changes.
Another problem related to the automation tools and the product is the way of inserting test data into the software. The interviewee mentioned that there is a high risk of breaking things when inserting test data into a running system. This is due to the fact that there is a lot of interaction between the data. It might even be impossible to change some things that have been inserted into the database during installation. Right now, they try to move the test data to external XML files instead of working directly on the database. While facilitating the handling of test data, this could lead to scenarios that can not be tested at all or can not be tested in a realistic way. Interestingly, this problem was expressed with regard to automated testing only. Why the test data problem does not apply to manual testing is not clear from the interview data.
Finally, latency was mentioned as problematic with regard to automated testing. As it can happen that actions take a different processing time every time they are executed, it can be necessary to introduce waiting times. It is hard to determine these waiting times properly. Furthermore, it is cumbersome to determine realistic maximum waiting times for every action performed in the GUI.

Company D
The quality assurance team is mainly constrained by time issues which restrict them from doing more testing. In the waterfall projects, this means that they can not repeat their tests. Running through the manual test suite once is however successfully applied. Even though the tests are not repeated, they are still documented for reporting and traceability purposes.
In agile projects, it happens regularly that the test suites are too big to automate them completely within the given time. Therefore, there is need for prioritisation:

"So for example we had a project where we had around 1000 test cases. So we said first 150 are priority A, next 150 priority B, and the rest priority C. And our main goal will be to have priority A fully automised and then when we finish that we move to priority B and then we decided to make it compatible with other browsers and we didn't touch priority C for test automation."

Again, no problems with regard to manually testing the test cases with low priority were brought up. One interviewee stated that the problem was more the maintenance effort than the creation of the test cases:

"Because the problem is that once you have a high portfolio of test cases to manage you will discover that you lose a lot of time in maintenance and you don't have time to develop new ones."

Regarding tools, they have some limitations with Selenium WebDriver when it comes to functionality like drag and drop:

"We are developing web applications that are using JS-based frameworks and it's very hard to replicate complex user actions like drag n drops for some browser(s), for IE or Google Chrome."

Finally, it was mentioned that the quality assurance team often only has an informative role. That is, they inform stakeholders of the product's quality, but can not influence whether it is released or not.

Company E
The team stated that they do not have any tool problems. The Selenium recording functionality was discarded. But according to the interviewees, Selenium WebDriver is rather accepted within the team. However, they stated that it is accepted as "the lesser evil". This could indicate that there is still a lot of potential when it comes to tool support for automated GUI acceptance and system testing. The main issue regarding a higher test coverage is rather a time and investing problem according to their perception. Manual testing is not done at Company E apart from informal acceptance testing. Therefore, no problems regarding manual testing were mentioned.
Test data is a problem they mentioned. According to them, it is hard to have a "proper back end and no dummy data".
Finally, they brought up that there is currently a lot of resistance against testing using design-by-contract principles. They would like to try out exploratory testing instead and do acceptance-test-driven development with more customer involvement.
The team said that they did not have any traceability between their tests and user stories, but did also not need it so far. This was mostly because of the limited project times and small test suite sizes.

Company F
The main problem at Company F is lack of time. So far, there is no testing at all apart from manual exploratory system testing. According to one interviewee, this is mainly because in the past the company focused on getting the application released as fast as possible. Therefore, they focused on new functionality instead of testing or quality assurance.
Now, the application does not require so many new features any more. Therefore, they are starting to focus on processes and testing.
Generally, it was stated that the application is hard to test. Therefore, introducing tests now is difficult. However, this was mostly brought up with regard to unit testing.

"We have made some tries to do more unit testing with mock-up data. But the product is not designed for that from the start. So it's hard to do that now. It's not designed for testability."

The general need for formalising the testing process was stated by both interviewees a couple of times. This was not only mentioned with regard to GUI acceptance and system testing.
Furthermore, it was stated that so far there are very few written requirements. With regard to GUI acceptance and system testing this means that it is very hard to verify if requirements have been fulfilled or not.
Overall, it can be said that Company F expressed the need to formalise their testing - whether manual or automated.

4.1.12. Case study question 12

Are there any scenarios which can not be tested using the current tools?

Company A
Using Sikuli, Company A has been able to automate even complicated interactions with the GUI, such as rebooting the system under test and continuing the test case afterwards. So far, the only type of scenarios that could not be automated were scenarios which require physical interaction with the system.

"With the automated tests so far the only tests which we ran into are tests which require physical interaction with the system. For example one of the test cases at company A, you have to push a red physical button which is not connected to software. So there is no way for to the automated tools to click on it."

Manually, the company can test every test scenario.

Company B
The same experiences as in Company A apply to Company B as well.

Company C
The first team reported issues with diagram interaction. They said that there are some limitations in Jelly when it comes to working with diagrams, like drawing something. Furthermore, it could happen that the results could not be verified properly if they are related to diagrams, for instance if a diagram is correct or not. One interviewee stated:

"Some of the things like diagram stuff, visually draw something. There are some limitations in functional testing with diagram stuff. [..]. Sometimes you can not exactly locate where something is. And sometimes there is no functionality to handle that."

Furthermore, with the new version of NetBeans, there are some bugs regarding simple GUI behaviour, like clicking a finish button in a wizard:

"[..] NetBeans 7.1, that is the latest release. They have some issues with the Jelly framework. Some obvious things are not working properly. [..]. Clicking a finish wizard. The framework cannot identify the finish button properly so it will have a time-out exception."

They stated that for important test cases they can still test it manually. However, this is not desired by the team.
In addition to handling GUI components, it was mentioned that it can be hard to test things that are related to non-local parts of the product:

"Some things, historically, are still hard to have locally. The stuff in the process that goes somewhere else, shared locations, is hard to test [..]."

This issue applies both to automated and manual testing.
In the second team, data issues were reported as well. The reason for this is, according to one interviewee, that the overall product of Company C has "a large database where everything is connected". Therefore, it is not easy to add test data without interfering with other data. In addition to that, some data within the database is connected to a company, whilst some other data could be installation data. However, this issue was brought up with regard to automated GUI acceptance and system testing. How this problem is avoided in manual system testing was not mentioned.
Furthermore, there are issues regarding the WindowsAutomation framework in the GUI automation approach. As mentioned before, there is a problem that not all of the code has WindowsAutomation functionality. This leads to problems in using these parts of the GUI. Instead of identifying the controls directly, indices, or even screen coordinates have to be used. The interviewee stated that this problem occurred both in the company's own code and in third-party components used within the product.

Company D
The interviewees mentioned complex user actions as an example for things they can not automate. Here, they named drag and drop functionality as one of the things that they can not test.
Regarding manual testing, it was brought up that some test scenarios can not be tested because of data limitations. For instance, creating too much dummy data could prevent the test scenario from being tested.

Company E
The interviewees stated that data-related tasks can be problematic. However, this was mostly due to time and investing problems, not due to tool or other limitations.

Company F
Two different things were mentioned by the interviewees. One interviewee brought up a new service API that they implemented which can not be tested properly. It integrates with third-party software services. This integration is very hard to test.

"We have a new [..] API to integrate with [..] (service) centres [..]. And especially the integration [..] is hard to test because we don't have their software at our location."

The other interviewee stated that realistic test data is hard to generate. So it can happen that issues arise first when the end user uses the product.

4.1.13. Case study question 13

How is the test suite maintained?

Company A
Concerning the manual test cases, the maintenance is done during the requirements phase or during manual testing. The test cases are simply updated once a tester or developer realises that they are not applicable any more. However, the updates are usually minor.
In the automated test suite there has so far been no maintenance as the test suite has not been used in a life project, yet.

Company B
In both the manual and the automated parts of the test suite, the maintenance is done manually once it becomes necessary.
So far, maintenance of the automated test suite was done manually at Company B. As an example, images of the visual GUI testing scripts had to be replaced.
The maintenance costs of this manual maintenance have been estimated to be approximately one fourth of the development costs of the automated tests. This estimation has been done during a major refactoring of the system which caused a big part of the automated test cases to fail.

Company C
In the first team, the classic concept is used that a person who breaks the build fixes it. The automated tests are run during each build on the team's continuous integration server. If a test fails, the relevant person who committed the new code is informed. This person is then responsible for fixing it. One interviewee reported that they had some problems with automated tests constantly failing. They disabled them and some person had to focus on them. Furthermore, some tests which were constantly failing were removed. It was reported that before doing that, developers started to ignore the tests, because they were failing all the time anyway:

"I just recently cleaned up, removed some things that were not working. It's better to have the things that are working instead of getting it fail all the time. In the end we didn't care if it failed or not, but it's important to listen to it."

The manual tests in the second team are maintained by two developers. They get suggestions from other developers and update the test repository accordingly. As there is no automation here, this is a pure manual effort.
One interviewee mentioned with regard to the GUI automation project that he was concerned regarding the maintenance effort of the GUI automation and if it would be feasible to automate a big part of the process models:

"I mean we're feeling that it's too slow and we are too worried about maintenance that we don't have any aim to sort of automate all of our processes. We are going to have sort of a low-level automation. The most important stuff we are going to automate."

Company D
Test suite maintenance is done once a test fails. It is then checked if the failure is related to broken test code or to a bug in the product under test. If it is a bug in the product, the developers are informed. Otherwise, the test case is being fixed by one of the quality assurance engineers.
The manual test scenarios are not maintained, as tests are only run once.

Company E
Maintenance is done using the principle to fix the build once it fails. For this, the team has a continuous integration server notifying the developers once one of the automated tests fails. If this happens, the responsible developer fixes it. As the acceptance testing is done manually by the customer in an informal way, there is no need for maintenance here either.

Company F
As there is no formalised testing at Company F, there is no maintenance right now.

4.1.14. Case study question 14

Does the product run on different platforms?

Company A
Company A's system is a desktop system. However, it incorporates touch screen functionality and is intended to be used on systems with touch screens.

Company B
Company B's system is similar to Company A's system when it comes to the platforms. It is also a desktop system intended to be used with a touch screen.

Company C
The first team's product is a Java product. It is developed in Java using the NetBeans IDE. The team uses some third party common IO, but the product itself is only running on one platform.
In contrast to that, the second team works with the Microsoft .Net platform. The current version of their product uses version 4 of Microsoft .Net.

Company D
The products are web applications only. Therefore, they have to run in different browsers on different operating systems.

Company E
The focus of the company is on Java web applications. However, they also use the Microsoft .Net platform sometimes.

Company F
Company F's system consists of three parts: a desktop application, a mobile application, and a web application. The desktop and the mobile application are both only running on a single operating system. If the web application is supported by multiple browsers was not mentioned during the interviews.

4.1.15. Case study question 15

Which aspects are considered by stakeholders in order to determine project success?

Company A
The system developed by Company A is safety-critical. Therefore, there is a large focus on testing and quality in general. Of course, there are also a lot of time constraints. Therefore, it can be said that Company A tries to develop high-quality software within their constrained environment.

Company B
The things noted for Company A apply equally to Company B as well.

Company C
In the first team, a number of different aspects were mentioned concerning what is of importance to the users of their product. First of all, it was said that the developers would like to continue working in the way they did before. However, it was also clarified that they were usually open for feedback on how they could improve their way of working.
The second point was the speed of fixing errors in the product. According to one interviewee, the customers are happy when they get their bugs fixed within a short time:

"So our customers are quite happy when they see that bugs are being fixed within a week."

In contrast to this, it was said that the delivery of functionality is much more important than the quality assurance process. It was even said, that management or customers do not see the value of a focus on quality:

"We think that it will get us the stuff on time if we focus on quality, but they (management/customers) don't see that."

Within the second team, it was said that the main focus lies on quality. The product should be stable and not come with any bugs that have to be handled by other teams using the product.
Finally, for the GUI automation project it was stated that there is a clear need and pressure to have automated GUI testing. This pressure comes from different areas within the company. There is a goal for the department to automate GUI testing, testers are complaining that their current testing is tiresome, and finally the agile way of working leads to it as well as it requires developers to continuously test their software:

"It comes from all levels. It comes from management. There is a goal for (the department) to make GUI testing for this year. There are testers saying I don't want to test. It's boring. Can't you fix this? [..]. Agile is one of the things that are pushing this, because suddenly people need to test the same thing quite often. [..]. And then probably if they need to do the same stuff and it's working today [..] they get bored."

Company D
As the interviewed team is working on quality assurance, the success of the projects is related to the quality of the final project. However, the team has no influence on the actual release of the project according to the interviewees:

"For example from the waterfall projects I would like if the decision to go online would have been based on the test coverage and the remaining open bugs. But sometimes this is not in our decision. So we just inform the stakeholders of the actual quality of the product and it's their decision."

That is, the team then has an informative role.
They also stated that in some cases there are clearly defined target dates until which the quality assurance has to be finished. The success can then be easily measured based on if the team meets the deadline or not.
In general, the good communication between the quality assurance and the product and project management was mentioned by multiple interviewees.

Company E
The interviewees stated that there are a lot of differences from customer to customer. They

had customers which have been involved heavily with testing. Others only focused on the nice look and feel of the system. No interviewee could remember a case where a customer was really interested in their quality assurance process during the project. However, they stated that they had customers choosing the company again because of successful projects in a past. They brought up the possibility that quality might have been an important influence factor in these cases.

Company F
The company is an innovator in its area. There are not a lot of competitors with similar products on the market. Therefore, the main aspect is the functionality of the system. One interviewee stated that a lot of their customers can use this to gain new contracts.
Another important aspect was brought up by the other interviewee. He mentioned that the product has to be stable as it is used all day long by its customers.

"Well, they work 24/7, so they want a product that works 24/7, that won't break down. Stability is a big concern."

Both interviewees mentioned that the mobile application has to be easy to use. In their opinion, the users are often not very used to computers or do not like to use any software applications. Therefore, the application should not impose too much overhead on them.

"And the mobile clients should be easy to use. Because not all (users) are that computer literate."

4.2. Analysis of case study hypotheses

For the performed case study, nine hypotheses were developed as a part of the study's theoretical framework presented in 3.1. The case study questions which were answered in the preceding section aimed at supporting or contradicting these hypotheses. In the following, every hypothesis will be handled independently.

4.2.1. Hypothesis 1

GUI acceptance and system testing is conducted in an unplanned fashion.

System testing is conducted by all cases under study. However, Company F has only an informal system testing process and tests are not documented. Company D and Company E use Selenium WebDriver for system testing. The tests are run repeatedly on continuous integration systems. In their waterfall projects, Company D uses manual testing. The tests are planned and discussed before they are conducted. Company A, B, and C use both manual and automated system testing in regular, planned intervals. Company A is in the process of introducing automated GUI acceptance and system testing.
Regarding acceptance testing, Company A, B, and D contradict the hypothesis. Acceptance testing plays an important role there and the systems are tested through the GUI, both manually and automatically using Sikuli. This is done regularly in a planned way. However, Company A and Company B are developing safety-critical software. Therefore, there is more need for thorough testing than in the other companies where this is not the case. Company D does manual or automated acceptance testing based on the process they are using. Company C has no acceptance testing of any form within the studied teams. Company E and F have a form of acceptance testing, but this is no formalised or documented process.

4.2.2. Hypothesis 2

Acceptance test scenarios are only elicited by developers.

At Company A and Company E, the test scenarios are brought by the customers. In Company B, the customer is involved in the acceptance testing process, but the scenarios generally come from legacy according to the interviewee's knowledge. Therefore, it can not be evaluated if the test scenarios were originally elicited by or with customers. In Company C, there are currently no acceptance tests. However, it was brought up that it would be desirable if the company's business analysts could write automated acceptance tests in the future. In Company D, the quality assurance department is responsible for the acceptance testing. They also elicit scenarios from a number of documents. In Company F, there was acceptance testing in the form of beta tests conducted by some of their customers. In this case, it would also be the customers eliciting the test scenarios for the beta testing. Whether this beta testing is still done is not clear from the interview data.

4.2.3. Hypothesis 3

Customer involvement for testing purposes does not exist or is at a minimum.

For Companies C, D, and F this is true. However, the reasons are different. In Company D, the quality assurance department was interviewed. If there is any customer involvement within the projects in general can therefore not be answered. In Company C, the interviewed teams have almost only internal customers. With these, an informal communication is possible. In Company F, there is almost no testing in general. Therefore, there is also no customer involvement in testing apart from beta testing they had in the past.
In Companies A, B, and E, there is large customer involvement, especially in testing.

4.2.4. Hypothesis 4

Test suites are designed in a linear fashion. Test cases are not connected.

This statement is false in most of the studied cases. Even though all cases apart from Company B had linear test cases, numerous times the interviewees stated that their test cases are connected logically. That is, complete test cases form preconditions to others and have to be executed first. At the same time, interviewees in Company C stated that this can be problematic. Additionally, it was common to use typical software development mechanisms, such as inheritance, to reuse reoccurring parts of automated test cases.
Company B has interconnected tests. They use a structure where test scenarios are broken down into smaller pieces. These can then be shared by multiple test cases.
The hypothesis can not be evaluated for Company F as no testing exists so far.

4.2.5. Hypothesis 5

Popular automated GUI testing tools are FitNesse, FIT, and Selenium.

This hypothesis proved to be mainly false. The only two companies which use Selenium are Company D and Company E. They use the programmatic interface Selenium WebDriver. The other four companies use different tools or no tools at all in the case of Company F. Two interviewees at Company E stated that FitNesse had been used in some project but was now not used any longer.

4.2.6. Hypothesis 6

Maintenance of test suites is problematic.

Company B seem to contradict this hypothesis. So far, they estimated that approximately a fourth of the cost for test development is needed for maintenance. In Company A, there are not yet any experiences with test suite maintenance for their automated tests. However, maintenance of the manual test suite was not described as problematic by the interviewee. Within one of the teams at Company C, there have been some problems with tests failing repeatedly. However, it was not brought up as a major problem. Again, no maintenance problems with respect to the manual test suite in the second team were brought up. Company E did not report any maintenance problems either. This is mainly due to their small test suite sizes for system testing. Both Company C and Company D mentioned that maintenance for automated test suites will become a problem once the test suites start to get big. Therefore, Company D does prioritisation. Within Company C, it is also planned to automate only tests for the most important parts of the system. For Company F, the hypothesis can not be evaluated as formalised testing does not exist.

4.2.7. Hypothesis 7

There is a clear need for improvement of GUI acceptance and system testing practices.

Similar to some of the earlier hypotheses, Company A and Company B contradict this hypothesis. The current approach applied in both cases looks promising so far. Especially in Company A, there is more experience needed, though. Company E did not strongly state that there is need for improvement. However, they wished for more acceptance testing with stronger customer involvement. Similarly, Company D did not state the need for improvement in tools. However, they wished to have more influence on the release process. Furthermore, the need for improvements with automated GUI acceptance and system testing can be seen in the fact that they have to prioritise test cases at the moment. Without this prioritisation, they would not be able to maintain all of their automated test cases. Company C stated the strong need for improvements. Manual testing was seen as a problem as it gets tedious and as it is slow. However, current tools for GUI test automation come with high maintenance costs which are not feasible for a high degree of automation. Furthermore, their process does not clearly support testing at the moment. Company F has no testing of any form. They would like to improve their testing practices in all areas, if possible.

4.2.8. Hypothesis 8

GUI testing in general and GUI acceptance and system testing in particular does not play an important role in industry.

This hypothesis is false with respect to the studied cases. Especially manual system testing on GUI level plays an important role in the studied companies. All studied companies have a form of manual system or acceptance testing on GUI level. However, acceptance testing with customer involvement is done in very different ways. Company D is performing beta testing which can be seen as manual acceptance testing by the customer. Company F did the same in the past. Whether or not they still conduct beta tests is unclear. In Company E, only informal acceptance testing by the customer is done after each iteration. Even though formalised acceptance testing is not done by some of the companies, it was still brought up as desirable by most of the interviewees. Therefore, it can be stated that acceptance testing does not yet play an important role in most cases, but the need for it is seen.

4.2.9. Hypothesis 9

The reasons against a wide adoption of GUI acceptance and system testing are mainly missing customer interaction, missing tool support, and large overhead of test case creation and maintenance with the existing tools.

At Company C, this hypothesis is true with respect to all three points. Missing customer interaction or too little customer interaction was mentioned by Company E. Company D did only bring up large overhead of test case maintenance for automated testing. However, they have adopted manual GUI system tests. The automated approaches at Company A and B look promising, but so far too few experience exists to properly evaluate the outcome of it. However, manual GUI acceptance and system tests have been established there for a long time. At Company F, there was so far simply no time to do testing.

4.3. Summary of case study findings

The case study findings are the answers to the case study questions and hypotheses. These can be summarised as follows:
In the studied cases, there is more testing on system-level than expected prior to the study. This is mostly conducted in the form of manual system testing through the GUI. Customer interaction does exist, but only rarely for testing purposes. In these cases, it is mostly non-formalised, improvised testing with no defined or documented test cases.
In most studied cases, test scenarios are elicited by developers or testers only. Often, vague scenarios, like user stories, are extended to full test scenarios by the developers or testers. Automated GUI acceptance and system testing is done, but various problems exist. Mostly these are attributed to missing time, problematic handling of test data, and missing tool support. Maintenance of manual test suites is working well within the studied companies. However, interviewees in two of the studied cases stated that maintenance of automated test suites will get problematic once many test cases exist. However, maintenance of existing automated test suites seems to be efficient. This can be attributed to a well-structured test suite as in Company B, or to test suites with few test cases, as in Company E. GUI acceptance and system test suites usually consist of linear test cases. Logical connections between test cases are often used. However, they can pose problems as coverage is hard to estimate and errors are difficult to find if one of the first test cases in an execution chain of several tests fails. An overview of the hypotheses and which cases confirmed or contradicted them is given in table 4.1.

The existing approach at Company B looks promising. They have extensive GUI accep-

	Hypothesis						
Company		A	B	C	D	E	F
	1	-	-	X/-	-	X/-	X/-
	2	-			-	-	
	3	-	-	X	X	-	X
	4	-	-	-	-	-	
	5	-	-	-	X	X	-
	6		-	X	X	-	
	7	-	-	X	X/-	X/-	X
	8	-	-	-	-	-	-
	9			X	X/-	X	-

Table 4.1.: Hypotheses (X for supports, - for contradicts)

tance and system testing with interconnected test cases. An overall graphical structure depicts the connection between test cases and facilitates maintenance of the automated test suite. On the other side, especially Company E seems to manage very well with few test cases and a lightweight GUI acceptance and system test suite. Here, only the most important scenarios are implemented as automated GUI system test cases. Acceptance testing is done manually by the customer in retrospective meetings. However, a broad code coverage is reached at unit testing level in order to find bugs.
These are essentially the two extremes found during the case study. Therefore, two different levels of GUI acceptance and system testing are proposed in the framework which is presented in the next section.

4.4. The PASTA Framework

The information obtained from the case study shows problems as well as approaches in the area of GUI acceptance and system testing. From this information, a framework is derived here which is intended to facilitate GUI acceptance and system testing. Based on its purpose, it is called the Planned Acceptance and System Testing Adoption framework, or PASTA framework.
During the data collection, essentially two different levels of GUI acceptance and system test suites were found. On the one hand, there were teams with organised-level test suites. These contain many test cases and require a planned and structured approach. On the other hand, there were teams which only did improvised GUI acceptance and system testing or simply had very few test cases. These two levels of GUI acceptance and system testing are depicted in figure 4.4. Additionally, a project could be in the process of transitioning from one level to another. This transition can be the result of a decision to transition (explicit transition) or due to ongoing changes in the test suite (implicit transition). Finally, it might be possible that the project does not fit into the PASTA framework. An example for this are the waterfall projects at Company D. Manual system testing is performed successfully in these projects, but tests are not repeated. Furthermore, the need to repeat or automate them in the future is not expressed. Such projects require different approaches and do not need a specific structure or architecture in order to succeed.
There are a number of different factors which affect where on the range between a base-level and an organised-level test suite a test suite is. These factors can be used as indicators for which kind of test suite should be used in a project. However, not all of them need to be present in a project in order to be in one of the levels.

In the following, the two test suite levels are discussed in detail. The levels contain guidelines for the four different perspectives of test suite design, test scenario elicitation, test automation, and test maintenance. Furthermore, explicit and implicit transition between the two levels is discussed.
The guidelines follow from the information gained in the case study. Furthermore, current research presented in the related work chapter of the thesis report is partly used as well. Finally, some guidelines are implications from the observed projects in the multiple-case study.

4.4.1. Base-level test suite

A number of project factors can indicate the use of a base-level test suite. These are depicted in figure 4.2.
The most important factor is that the product is not safety-critical. This means that there is no legislation which requires acceptance or system testing coverage for all requirements

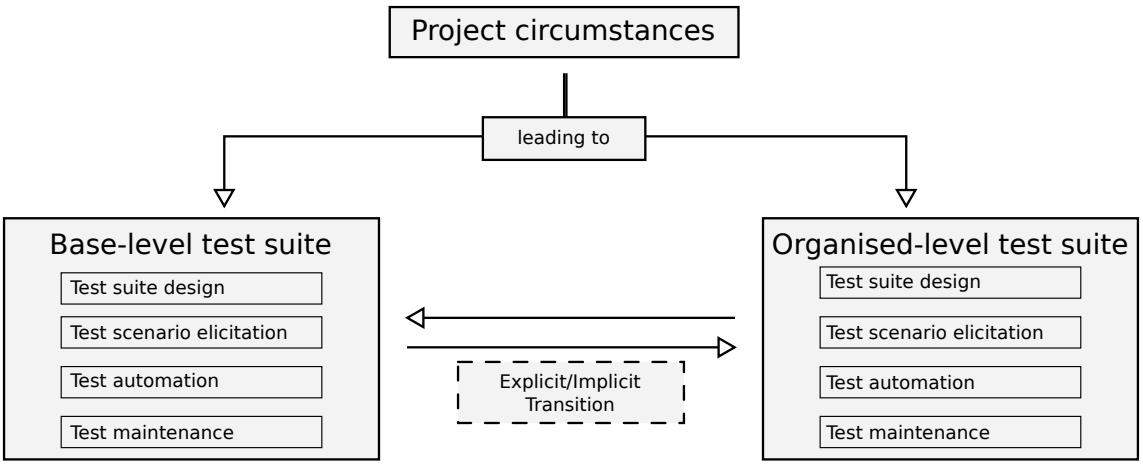

Figure 4.1.: PASTA framework overview

or for as many requirements as possible. All studied projects which used base-level test suites were not safety-critical. Additionally, time issues can lead to a base-level test suite. This covers both the duration of the project and too little time and resources in general. In the studied cases, Company E generally had projects with a duration of less than a year. Therefore, it was not feasible for them to create and maintain many system tests. Instead, they tried to cover the most important test cases within their limited time. The first team at Company C did not have enough resources and time in order to create more tests. However, they did not state any serious quality problems arising from their practice of covering only the most important functionality with their automated system test cases. If GUI acceptance and system testing shall be introduced incrementally, a base-level test suite is a good choice. The test cases can then be gradually added until a more sophisticated test suite is needed. Only then a transition will be necessary. Introducing an organised-level test suite from the start would come with high costs and require a large investment of time and resources. Therefore, starting with a base-level test suite is an obvious decision for smaller projects or smaller companies. Especially in agile projects, it is necessary to start with a base-level test suite. An organised-level test suite would impose too much overhead on the project. One interviewee at Company E stated generally that it is always possible to have better and more testing with more resources. However, investing this time or the resources was usually not possible. The interviewee acting as a proxy for Company A and B mentioned the same aspect. This aspect of incrementally introducing a concept instead of introducing it all at once has been discussed in many areas of software engineering, for instance in the area of Software Product Lines [51, Chapter 20].
Customer interaction was stated to be important by two out of five companies. In Company A and B, the interviewee could not answer if this was the case or not. However, the customer is involved at both GUI acceptance and system testing within these companies. Therefore, it can be assumed that customer involvement plays an important role there as well. If customers are not yet involved in testing, it is important to get the involvement started on a basic level. A base-level test suite is a good opportunity for this.
Another factor indicating a base-level test suite is a low priority for quality. If customers, management, or other stakeholders do not explicitly express the need for high quality and sophisticated testing, it is important to start with a base-level test suite. Otherwise, the great effort of building up and maintaining an organised-level test suite can not be justified. Interviewees at both Company C and E mentioned this aspect.
Finally, an existing GUI acceptance or system test suite with maintenance problems can lead to a base-level test suite. By re-evaluating the scenarios and refactoring the test suite,

maintenance problems could be reduced.

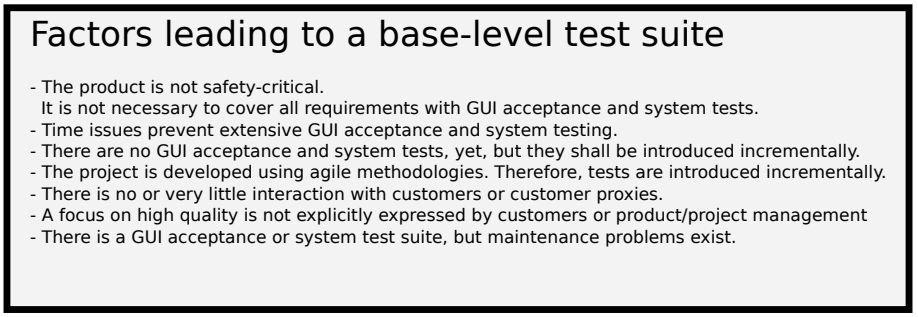

Figure 4.2.: Factors indicating a base-level test suite

The guidelines for base-level test suites cover four different perspectives. These are depicted in figure 4.3.
Concerning the test suite design of a base-level test suite, there are four important points. Firstly, GUI acceptance and system testing aims at validating or verifying if requirements have been met or not. Often, the validation of requirements and the finding of bugs overlap. However, finding as many bugs as possible with GUI acceptance and system tests should not be the main goal. Therefore, code coverage aspects are not central. This is also stated generally for GUI testing by Atif Memon in [42]. A high code coverage with the goal of finding errors or bugs should instead be aimed at on lower testing levels like unit testing which can be automated easily. The test suite should instead aim at covering the main important aspects of the system. This approach is successfully applied by Company E where the overall testing levels have a pyramid structure. A large code coverage is reached with unit tests whilst only a few system test cases exist. Behaviour-driven development, as introduced by Dan North in [49], follows a different approach. Here, user stories and scenarios follow a template and contain keywords which are mapped to the method names [56]. These user stories are then directly testable. The resulting test cases are then essentially system test cases which describe the *behaviour* of the system. However, the whole development process has to follow this idea of modelling behaviour. It is therefore seen as too heavyweight by the author to incorporate it into a framework on GUI acceptance and system testing. Furthermore, it would require a substantial overhead to introduce Behaviour-driven development into an existing project.
The second point for test suite design is to focus on the main scenarios. As only a few test cases should exist on GUI acceptance and system testing level, the focus should lie on the most important scenarios.
Thirdly, imposing additional complexity on the test suite by connecting the test cases is unnecessary in a base-level testing approach. Instead, the test cases should be kept unconnected. This also includes logical dependencies like test cases being the precondition to others. These dependencies were stated to be difficult by both teams in Company C. The practice at Company E shows that it is possible to maintain a base-level test suite successfully without a large structuring overhead.
Finally, additional overhead in the documentation should be avoided. Therefore, traceability from test cases to requirements is not needed. At this complexity level, developers should be able to see the connections without explicitly documenting them. For instance, this is the case at Company E and in the first team at Company C. However, if traceability can be added without a substantial effort, it could be done anyway in order to facilitate a potential transition to an organised-level test suite later on. This will be discussed in further detail in the description of the transition between the test suite levels.

For test scenario elicitation, there are three points. Firstly, customer interaction plays an important role. Company A and B use GUI acceptance and system testing with customer interaction successfully. Company C and Company E also mentioned the importance of customer involvement. In Company C this involvement is missing. However, they stated that it would be desirable to have it. Even though it is not possible to cover all possible scenarios, customer involvement will give the developers the assurance that they cover the scenarios which are most important to the customers. Without a customer eliciting the scenarios, the team can not know which scenarios to elicit, how exactly they should look like, and when to stop implementing tests. One interviewee at Company C stated: "How can you verify it's working if you don't know how it's supposed to work?". However, the interviewees generally reported that it is very hard to get the customers involved. This is also reflected in a study by Hoda et al. where customer involvement is called "one of the biggest challenges" within agile software development [30]. Even though it might be easier to involve the customer into the test scenario elicitation in other software development processes, it will still be challenging. A base-level test suite might be a good start for customer involvement as it will start with few scenarios only. Other options on how to convince customers can be found in the study by Hoda et al. [30].
As the main functionality shall be covered with tests, it is important to focus on positive scenarios. It is not feasible to cover all possible branches of a scenario, like exception and error handling. These parts of the source code should again be focused by lower testing levels like unit testing. Finally, it should be avoided to elicit scenarios without the customer. This should only be the last step if it is absolutely impossible to get customers involved. The reason for this is the focus of the base-level test suite. As a wide coverage is not intended, it is not feasible to elicit more than the most important scenarios that are elicited together with the customer.

Test automation plays a key role in agile software development methodologies, such as Extreme Programming [10]. It was stated by interviewees in Company C that testing functionality repeatedly gets tedious. Furthermore, it was stated that test results start to be ignored if they have to be repeated all the time. Therefore, automated tests should be written if an agile development process is used. For traditional software processes like waterfall projects, manual GUI acceptance and system testing can be sufficient.
A possibility for automated testing that can be used by both technical and non-technical people are visual GUI testing tools. It was stated by interviewees in Company C that it would be good if non-technical people like business analysts could write the acceptance tests. In Company E, they brought up that acceptance-test driven development could be a good way. For this, the customer needs to write the tests. This could also be enabled by visual GUI testing. Initial support for the use of visual GUI testing tools like Sikuli for system testing is also given in [16]. However, it can be expected that non-technical people can only write basic test cases with current visual GUI testing tools as the implementation of complicated scenarios often requires a deeper technical knowledge of the testing tool.
It is very important that testing is a mandatory practice. If it is only done ad-hoc when there is time left, it is easily avoided and the quality of the test suite suffers. Furthermore, it could happen that test cases which are hard to implement are simply ignored even though they are important. Similar problems with testing practices were found at the first team in Company C.
Finally, it is a good idea to use automated test cases as means of documentation if they can be recorded or visually shown to someone. This reduces additional documentation and relates to the principle of reducing waste in agile software development or to only have *sufficient documentation* as covered in [18].

Test maintenance should be done once it becomes necessary. With a small amount of test cases which are not connected it is not necessary to do any planned maintenance. In the case study, a common practice was to use a continuous integration system for automated tests and fix the tests once they fail. Test automation without a continuous integration approach was not found at any company. This reflects the finding of [19] that test automation without continuous integration is problematic. Maintenance problems for manual test suites were not reported in the study. Therefore, maintenance is uncritical if GUI acceptance and system testing is conducted manually.

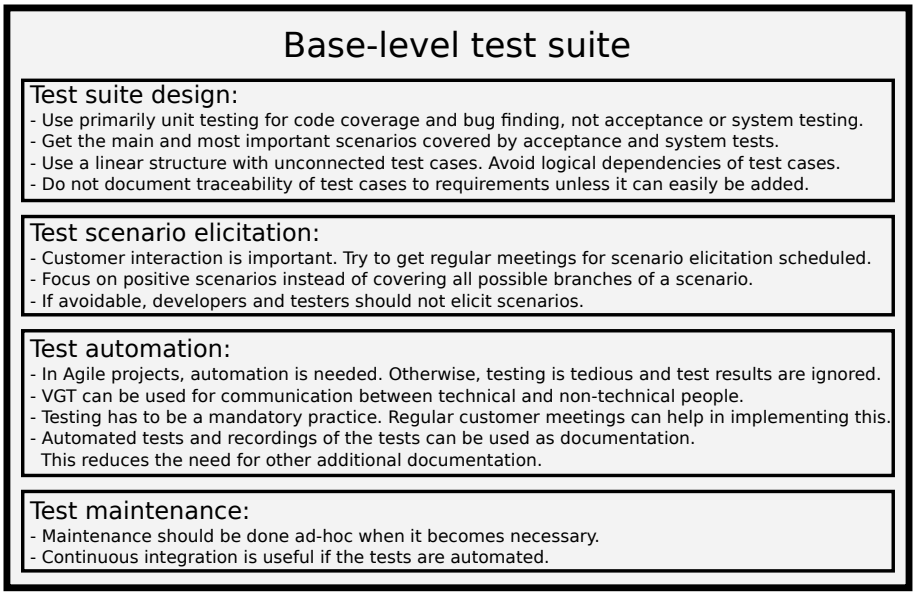

Figure 4.3.: Base-level test suite characteristics

4.4.2. Organised-level test suite

A number of project factors can indicate the use of an organised-level test suite. These are depicted in figure 4.4.
In contrast to the base-level test suite, a safety-critical product is a strong indication for the need of an organised-level test suite. Safety-critical software, like in Company A and B, requires a high coverage of requirements with acceptance and system tests.
For non safety-critical projects, it is possible that a strong focus on quality is expressed by stakeholders. In this case, an organised-level test suite makes sense. The second team in Company C can be seen as an example for this focus. It was mentioned by an interviewee that quality is very important for management. Furthermore, they already have a high coverage with their 150 manual system test cases.
In order to benefit from an organised-level test suite, the project needs to run for a long time. If the project's duration is too short, the effort to structure the test suite and establish the testing process is too high. The projects in Company A, B, and the second team in Company C are good examples for long-running projects where an organised-level test suite can be beneficial.
It can also happen that a project already uses GUI acceptance or system testing which shall be adapted to the guidelines shown below. Especially if a decision is made to refactor or extend the existing test suite this can be beneficial. The decision to automate an existing test suite, like the test suites in Company B, is also related to this point.
If customers or customer proxies are already involved in the testing process and see the benefits of their involvement, it is a sign for an organised-level test suite. If the customers see the benefits of GUI acceptance and system testing and understand the need for their

involvement, it is possible to get them involved even in bigger test suites.
Finally, there can be a plan to introduce extensive GUI acceptance or system testing. If a lot of functionality shall be covered, it is required to use the guidelines for an organised-level test suite. Otherwise, there will most likely arise maintenance or time problems with the test suite.

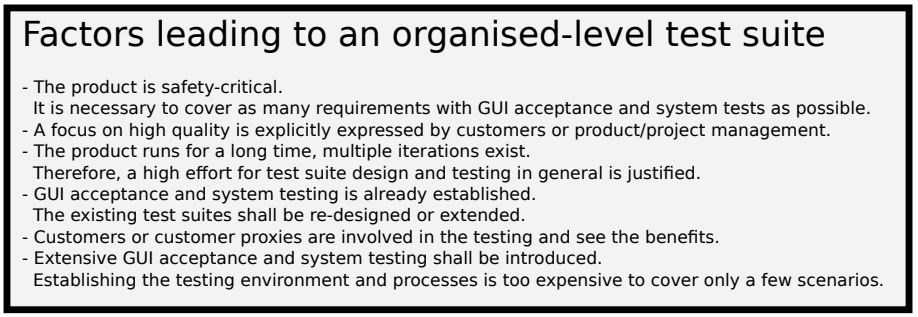

Figure 4.4.: Factors indicating an organised-level test suite

The guidelines for organised-level test suites cover four different perspectives. These are depicted in figure 4.4.2.
Concerning the test suite design there are four main factors. An organised-level test suite aims at covering a large part of the requirements with test cases on system level. In contrast to the base-level test suite, as many requirements as possible should be covered. As such a test suite will soon become very complex, it is important to keep a global view on the test suite. This includes traceability of test cases to requirements, grouping of test cases, and maybe even a logical structure with overlapping parts within the test suite. This depends whether the system is build in a way that overlapping tests make sense. For Company A it was stated that test cases are unconnected as the system functionality is very separated from each other. In this case, connecting test cases is not indicated. Using single test cases as a precondition to other test cases showed difficult in Company C. Preconditions made it hard to interpret test results and find bugs. Company B uses diagrams depicting the overall connections within the test suite. This enables them to reuse test artefacts and therefore share them among several test cases. First estimations suggest that this is a good way to enable reuse and lower maintenance costs for their automated test suite. Even if the system consists of very unconnected parts, like Company A's system, a global view on the test suite is important to not lose track of the system scope. This is related to the suggestion of Atif Memon in [42], to always keep a global view on GUI test suites in general. The global view is also essential for manual test suites as it is not concerned with any automation aspects.

Similarly to the base-level test suite, the customer has to be involved. At least, the main important scenarios should be elicited by or together with the customer. Some of the factors described earlier are concerned with customer involvement or a focus on quality. Therefore, it is likely that the customer can see benefits of his involvement and can therefore be persuaded to invest more time into a better test suite. As an alternative, it can be tried to involve other stakeholders with sufficient domain knowledge instead of customers. This is sometimes done in Company C. Other alternative options are discussed in [30].
The scenarios themselves should be described in an abstract way. That is, they should be robust to changes within the GUI or the system architecture. The idea behind this is that scenario descriptions should be kept in a format which requires a low maintenance effort but is still understandable to developers or testers. Atif Memon uses this concept for

naming single test cases as *tasks* in [43]. The concept of using simple *user stories* which are then extended by the developers also works well in Company E. Furthermore, abstract tasks are easier to reuse among several test cases. With less abstraction, this will get more difficult. The concept of shared tasks among several scenarios is also described in [8] for scenario management in general, where these shared tasks are called *episodes*.

Most of the interviewed companies either had automated test suites or wanted to automate their manual test suites. Reasons for this are that the testing phases are too short, repeated testing tires the employees, or the need for shortening the testing intervals. Especially in agile projects where regression testing is a heavily used, automation is inevitable. The testing tool should be chosen depending on the product characteristics. Visual GUI testing is useful if the GUI evolves steadily and slowly. Furthermore, products which are developed in parallel on multiple platforms might require visual GUI testing. Company B uses visual GUI testing successfully and in Company A it is being introduced at the time of this thesis project. If the GUI evolves or changes quickly, testing tools which identify GUI controls through their variable names might be more suited. The first team in Company C uses the Jelly framework rather successfully. For web applications, Selenium WebDriver is used by Company D and E. They stated that they are satisfied with the choice.
In order to enable reuse of tasks or similar concepts used to describe the tests scenarios, it should be tried to capsule these entities in single software artefacts. In the studied cases, typical object-oriented approaches such as abstraction and inheritance were used to do so. More important is that there is a clear practice on how this abstraction is handled. If abstraction is handled in an unplanned manner it will only complicate maintenance and not lower the maintenance effort. This concept can again be transferred to the idea of only having sufficient documentation in agile projects [18]. Creating abstraction is not the primary goal of the scenario documentation. Therefore, an overly abstract scenario description will only complicate the understanding and not aid the overall quality of the test suite.
Test data was mentioned during the interviews as one of the main problems. It is difficult to find a general solution for this, as the system data is stored and handled differently in almost every system. As a general guideline, the test cases should try to abstract from test data as much as possible. Simple tests, such as if a method returns the right value or not, should again be tackled by lower testing levels such as unit testing. Additionally, the test system should be fixed. The tests do not need to run successfully on every possible installation of the software! This also applies to other things such as customisation and personalisation of the GUI.

In the studied cases, continuous integration servers like Jenkins are widespread and accepted. Especially if there are no defined testing phases, it is a good idea to run the GUI acceptance and system test suite regularly. Depending on the size of the test suite and its execution time, the tests do not have to be run after every build. Weekly or even monthly test runs might be sufficient. For purely manual testing, continuous integration is not possible. However, the test suites still have to be run regularly.
The point of time when test maintenance is done depends on the used software development process and on the kind of maintenance that is necessary. If only single test cases fail and can be fixed easily, it is feasible to do so once the failure occurs. This is a common practice at Company C, the agile projects at Company D, and Company E. If changes in the system lead to a high number of test failures, it might be indicated to schedule special maintenance phases. Company B experienced this when they re-factored large parts of the system and caused almost all test cases to fail. Bigger maintenance problems with manual test suites were not reported. However, large changes in the system under test

might invalidate large parts of the test suite even if the descriptions are kept abstract.
If tests start to fail very often and the ongoing maintenance costs become very high, it might be a indication to re-evaluate test suite level decision. Maybe the implemented structure was not abstract or the test suite level is simply not feasible for the project. These aspects are discussed in more detail in the following section.

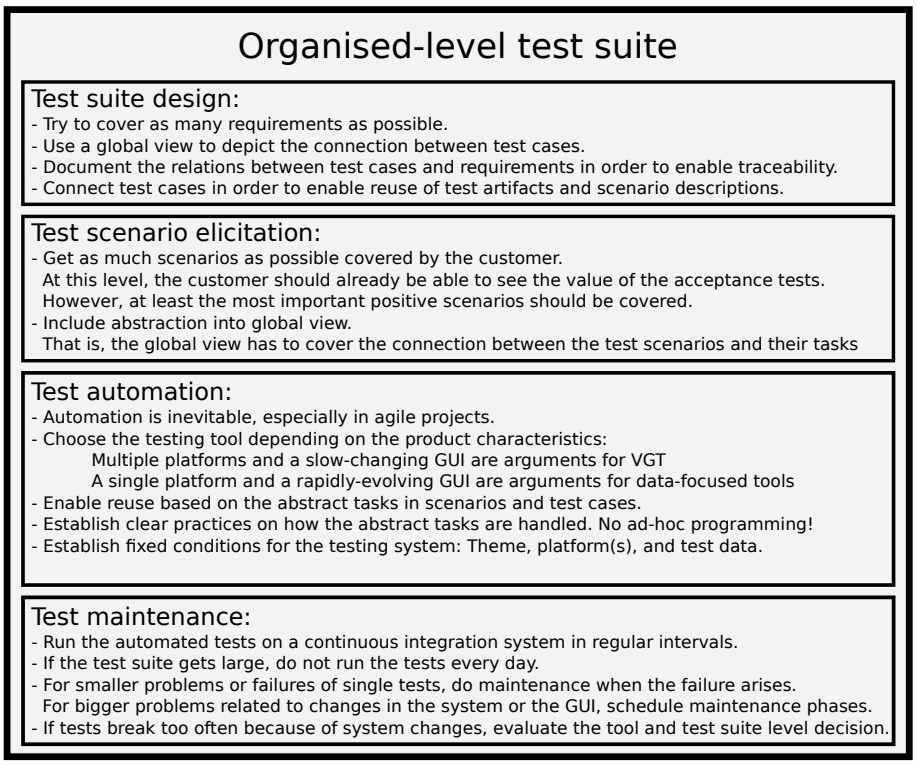

Figure 4.5.: Organised-level test suite characteristics

4.4.3. Transition between the levels

Test suites are usually not static. As the system evolves, so should the test suite. This can lead to a transition from one test suite level to the other. Regarding the multiple-case study conducted in the course of this thesis project, it is important to note that no transition was observed. Therefore, the following guidelines are logical implications. They take into account both the situations and problems in the studied teams.
There can be both implicit and explicit transitions. An implicit transition happens when the test suite slowly evolves into a test suite of another level. This happens without any decision, for instance by adding additional test cases and introducing abstraction and traceability in the test suite. On the other hand, there might be an explicit decision to transition. This could happen due to quality problems or due to time problems.
Two reasons for explicitly transitioning from a base-level test suite to an organised-level test suite were identified by the author. Firstly, a higher test coverage can be required by stakeholders or by the team itself. One reason for this could be quality problems with the software system. Secondly, it could happen that the existing test suite is extendible without investing a lot of time into it. If the test suite has been organised in an abstract way and aspects of the organised-level test suite were already respected during the introduction of the base-level test suite this might be the case.
Implicit transition from a base-level test suite to an organised-level test suite is a normal evolution aspect. By continuously adding new test cases the need to re-structure the test suite and introduce abstraction and reuse will eventually arise. Especially agile projects

face this problem as automated test cases are slowly added incrementally as the system grows. The first team at Company C can be seen in this stage.
In order to succeed with the transition, it is important to introduce the structuring concepts presented in the organised-level test suite section. Connections between test cases and reuse of test artefacts is needed in order to reduce the high maintenance costs. Traceability needs to be introduced in order to not lose track of the test suite. Company B faces this problem. Many of their test scenarios come from legacy and it is now no longer possible to trace them to requirements.
If the decision to transition is not made or influenced by customers, it is important to convince them that additional commitment from their side is necessary and beneficial. Furthermore, if an implicit transition is noticed the choice to stay in the lower (or higher) test suite level can be advisable. Especially if additional resources can not be invested in the testing process this is a natural step.

The transition from an organised-level test suite to a base-level test suite is also possible. However, it is usually associated with problems in the existing test suite. A reason to implicitly transition from an organised-level test suite to a base-level test suite is that test cases are increasingly discarded because they fail on a regular basis. This happened with some test cases in Company C. Instead of fixing test cases they are increasingly ignored and finally discarded. If this is an ongoing process, it can happen that only the most important test cases remain in the end. Explicitly, there are a number of different possible reasons for transition. Firstly, it can happen that it was simply not successful to introduce an organised-level test suite. For instance, the chosen abstraction and structuring concepts did not help in keeping the maintenance costs low or the costs were simply higher as expected. This might happen when the test suite is very connected even though the system is not structured in such a way. The interviewee at Company A brought up this aspect. As the functionality in Company A's product is very spread out, connecting test cases is not beneficial and would only add extra overhead. Secondly, resources and time which can be spend on the test suite might not be sufficient in order to maintain an organised-level test suite. In this case, the decision has to be made whether the test coverage is more important or the saving of resources.
If the transition from an organised-level test suite to a base-level test suite is done, it is important that test cases are not simply removed and everything else is kept as before. The additional overhead for abstraction and organisation of the organised-level test suite has to be reduced as well. Therefore, connections between test cases and other dependencies between them should be removed. Furthermore, keeping a global view on the test suites is not needed any more.

5. Evaluation

The PASTA framework was statically evaluated at Company C, D, E, and F. For Company A and B, no evaluation was possible due to time limitations. In the following, the evaluation design is discussed. Afterwards, the evaluation results are discussed.

5.1. Evaluation design

Based on the PASTA framework, the current state-of-practice regarding GUI acceptance and system testing was evaluated for every company by the author. That is, given the observed data, the company was placed in one of the two test suite levels described in section 4.4. A report was handed out to Company C, D, E, and F describing the test suite level at which they were placed and which factors the decision was based on. Additionally, recommendations with respect to GUI acceptance and system testing were included specifically for these companies or specific projects within the companies. The recommendations were made based on the relevant test suite level of the PASTA framework and the company or project circumstances. Additionally to this report, the companies were provided with a document describing the overall PASTA framework. This report is similar to section 4.4 and was aimed at making the test suite level decision and the recommendations clearer.
The current state and the recommendations were explained to Company C, D, and F in a meeting and discussed with the participants afterwards. Company E did not find time to schedule such a meeting. Additionally, every interviewee at Company C, D, E, and F was asked to fill out a short evaluation survey. The survey contained questions regarding the interviews and their connection to the framework design. Furthermore, the current test suite level and the given recommendations were covered. The complete survey can be found in appendix B. The survey was conducted using a free online survey service.

5.2. Evaluation outcomes

In Company C, four surveys were handed in. That is, one interviewee did not hand in a survey. In Company D and F, the number of surveys handed in is equal to the number of interviewees. In Company E, four surveys were handed in for a total of five interviewees. However, from the survey comments in one survey it became clear that it was filled in by more than one person. Therefore, feedback from all interviewees exists for Company E as well.
In the following, the outcomes of the static evaluation are discussed for every company separately. They include feedback from both the feedback session and the surveys where applicable. Finally, a short summary on the evaluation outcomes is given.

5.2.1. Company C

At Company C, the framework was discussed in a meeting together with the contact person and some of the interviewees. Both the current state and the recommendations were explained by the author and then discussed jointly. In general, the framework and the recommendations were received positively. One of the participants noted that it is always interesting to get an evaluation from a person who is not employed at the company. The participants seemed to be especially interested in the *attitude towards testing* discussed in the data analysis part of this thesis.
One of the participants seemed to be unhappy with the tool recommendation. For the second team in the company, visual GUI testing tools were recommended for automated testing. This was due to the fact that during the interviews it was stated that the GUI of their product does not evolve rapidly. Additionally, they expressed problems with their current tool choice. However, during the meeting the participant stated that he was not sure whether visual GUI testing tools would be a good idea for their team. He mentioned that the GUI of their product can change rapidly. Therefore, using visual GUI testing could lead to even higher maintenance efforts than the current tools. This aspect of a rapidly-changing GUI was not brought up during the interviews. However, the fact that both visual GUI testing tools and tools which target the interaction between GUI components and the system are seen as problematic indicates that there is still more research needed in the area of automated GUI testing.
Four out of five interviewees at Company C handed in the survey. The full survey answers can be found in appendix C. As a summary, the main things to consider from the survey are as follows.
The connection between the interviews and the framework can be seen. Here, two participants agreed, one participant strongly agreed, and one participant was neutral.
The two proposed testing levels are seen as more or less suitable to describe most real-life projects. Two participants were neutral towards this. However, the other two agreed and strongly agreed.
The opinions on whether the framework can be applied in future software projects differ. Two participants were neutral, one disagreed, and one strongly agreed that the framework can be applied in future software projects.
Three out of four participants agreed that some recommendations do not fit their special case. The fourth participant answered neutrally.
While two participants were neutral on the question whether they would have placed themselves in the other testing level, the other two participants disagreed. Here, one participant stated in the comments that both of the test suite levels are needed in almost every project. However, the question should be about the balance between the levels. This is in contrast to the data gathered in the multiple-case study. Here, the teams which would have been placed in between the levels stated problems with their current test suite. For instance, the second team at Company C can be seen in between both levels. They stated that the current manual testing is tedious and automation is needed. However, they see heavy maintenance costs associated with the current automated GUI testing tools.
Interestingly, all participants stated that other factors are more important for placing a company in one of the two test suite levels than the factors stated in the report. They did however not leave any comments regarding any of these possible factors.
All of the participants agreed that they would try to implement some of the recommendations. However, only two agreed or strongly agreed that the cooperation was valuable for their project. The other two participants were neutral on this issue. The case study was rated with an average of 3.5 out of five stars.

5.2.2. Company D

At Company D, the framework was discussed in a meeting together with two of the three interviewees. The meeting was conducted using Skype including its video functionality. The participants generally agreed with most of the recommendations. However, they criticised the focus on unit testing for bug finding. They brought up the issue that in their company, a lot of bugs are found only on system level. They stated that this is due to the fact that unit tests can not test the integration of multiple sub-systems or components. This is a valid argument and should be considered in future work. Company E actually uses some integration tests in their overall testing process. It might be necessary to pay more attention to this layer of integration tests in future work.
Additionally, the participants mentioned that due to test-driven development, a lot of unit test scenarios are of a positive nature. That is, they only test the positive outcome of the developed functionality. Therefore, system tests should especially cover negative test cases and not positive ones as stated in the framework.
One of the interviewees mentioned that he looked at Sikuli and liked its concept. However, he would still prefer Selenium WebDriver for system testing of web applications.
At Company D, all three interviewees answered the survey. The full survey answers can be found in appendix C. As a summary, the main things to consider from the survey are as follows.
The connection between the interviews and the framework can be seen. Here, two participants agreed, one participant strongly agreed.
The participants are mostly neutral whether the PASTA framework is suitable to describe most real-life projects. Two participants were neutral towards this, while one participant agreed.
Two participants agreed that the PASTA framework presents solutions or ideas relevant for the projects they are involved in. One participant was neutral.
Two participants agreed that some recommendations do not fit their special case. The third participant answered neutrally. This might be attributable to the issues brought up during the meeting. That is, that the system tests need to focus on bug finding and on negative scenarios.
All participants disagreed to the question whether they would have placed themselves in the other test suite level of the PASTA framework.
Two out of three participants agreed that other factors than the discussed ones are more relevant to place a project in one of the two test suite levels. The third participant disagreed. Again, no comments were left on which factors might have to be included.
All of the participants stated a neutral answer to the question whether they would try to implement some of the recommendations. However, all of them agreed that the cooperation was valuable for their projects. The case study was rated with an average of 3.6 out of five stars.

5.2.3. Company E

Due to time limitations from company side, no meeting with Company E could be scheduled. Therefore, the factors relevant for the test suite level decision and the recommendations made for Company E could not be explained to the team. This might lead to issues regarding the understanding of the recommendations.
Four subjects participated in the survey. However, from the comments it can be seen that one survey was answered by more than one person. Therefore, it can be stated that all interviewees also answered the survey. Overall, the survey results for Company E vary greatly between each other. This is reflected both by the answers for each question but also by the comments left by some survey participants. The complete survey answers can be found in appendix C. As a summary, the main things to consider from the survey are

as follows.
The connection between the interviews and the framework can be seen. Here, three participants agreed, one participant strongly agreed.
The two proposed testing levels are seen as more or less suitable to describe most real-life projects. Two participants were neutral towards this. However, the other two agreed. One participant stated that he/she had seen "large, unorganised test suites which seem to really fit neither category". However, the participant also stated that this might be due to the fact that these test suites were "broken". Another participant commented that "it is actually possible to put projects into either category" and that "it feels like something is missing in between". A similar comment was already left at Company C. However, this might also be an understanding issue. The collected data suggests that these two test suite levels are indeed beneficial. While it is surely possible to place a project in between, the collected data suggests that this is not a desirable state. However, the issue has to be addressed in future work.
The participants mostly disagreed with the statement that the framework presents solutions or ideas relevant for the projects they are involved in. Here, one participant strongly disagreed, and two participants disagreed. Interestingly, one participant strongly agreed. This participant also commented that "Many ideas from the base-level side of the framework are very relevant" to his/her work. The participant strongly disagreeing with the statement commented that "no new ideas/solutions are presented". As the proposed framework consists of current practice in real-life projects and some practices in other research areas, this statement is true. However, the project's focus was not on coming up with new ideas or solutions but rather to investigate which existing ideas or solutions can be applied to GUI acceptance and system testing and how they can be combined. Therefore, the participant's expectations might have been different. Regarding this, the participant commented with respect to another statement that he/she "expected concrete advise on how to establish test suites in projects".
The opinions on whether the framework can be applied in future software projects differ. Two participants agreed, one disagreed, and one was neutral.
One participant agreed that the recommendations are valuable for their projects. Two participants disagreed, and one was neutral. One of the disagreeing participants stated that "the recommendations are mostly an enumeration of practices which are already in use in our company". This is true for some of the recommendations, as they were derived from interview data. As Company E is seen as an example for base-level test suites, this also indicates that good practices are recommended for this test suite level. However, additional recommendations were also made.
In contrast to Company C and D, the statement whether some recommendations do not fit the company's specific situation was answered mostly neutrally. Two participants gave a neutral answer, while one participant strongly disagreed and one participant agreed.
Three participants disagreed that the other testing level would have been more suited to their case. The fourth participant even strongly disagreed.
Whether the participants would try to implement some of the recommendations was answered neutrally by one survey participant. Two disagreed, and one agreed on this statement.
The participants were all neutral to the statement regarding whether the cooperation was valuable for the company or not. The case study was rated with 2.6 out of five stars.

5.2.4. Company F

At Company F, the framework was discussed in a meeting together with all interviewees, the contact person, and some additional employees interested in the results. The meeting was conducted in person at their site. They generally liked the work and the recommendations. However, they brought up some issues. Firstly, they stated that for testing,

requirements are needed. However, they do not have any documented requirements. This can be seen as an additional argument for the need of customer involvement. However, they also stated that this is difficult as customer involvement comes with expectations from the customer side. It would be difficult to sell to their customers an involvement with only better quality as an advantage. Usually, customers would expect that their requests get prioritised if they have an involvement with the company. Additionally, they mentioned that they have the legacy problem of focusing too much on customer requirements. Therefore, they have a lot of features in their system which only some customers required, but which are unnecessary. So, to get customers involved in general, it would be difficult to choose the right customer to involve. This can be seen as an additional problem for all market-driven companies and should be addressed in future work.
Both interviewees at Company F answered the survey. The complete survey answers can be found in appendix C. As a summary, the main things to consider from the survey are as follows.
One participant strongly agrees that the connection between the interviews and the framework can be seen. The other participant was neutral towards this statement
Both agreed that the two proposed testing levels are suitable to describe most real-life projects.
The opinions on whether the framework can be applied in future software projects differ. One participant agreed, the other one was neutral.
One participant gave a neutral feedback and one agreed to the statement that the PASTA framework presents solutions or ideas relevant for the project they are involved in. The answers were given to the question whether the framework can be applied in future software projects or not. However, the answers were inverted here.
One participant agreed that some recommendations do not fit their special case. The other participant answered neutrally.
Both participants agreed that the recommendations are valuable for their project. However, one participant agreed that some recommendations do not fit their case. The other participant stayed neutral here.
In contrast to Company C and D, both participant gave a neutral answer to the question whether other factors are more relevant for placing a project in one of the two testing levels. This might be attributed to their missing experience with testing in their company.
Both survey participants agreed that they will try to implement some of the recommendations and that the cooperation was valuable for the project. The case study was rated average four out of five stars by the survey participants.

5.2.5. Summary of the evaluation outcomes

The opinions on the PASTA framework, and regarding the individual test suite levels and recommendations differ from company to company and also between survey participants in some companies. Therefore, in order to validate the framework, more work is needed in the future. The main two issues which were mentioned by multiple survey participants are as follows.
Firstly, it was mentioned multiple times that the two test suite levels are not sufficient for real-life projects. Within the studied cases, mainly these two cases were found. Projects which did not perfectly fit one of the levels, such as the second team in Company C, reported problems with their current GUI acceptance and system testing practices. However, it might be possible to conduct GUI acceptance and system testing successfully on a level in between the two proposed levels. This has to be investigated in future research on the topic.
Secondly, it was mentioned multiple times that other factors than the factors proposed in the framework design are indeed more relevant for choosing the test suite level. One of

these factors might be the sort of customers the project has. It is possible that a market-driven company needs a different test suite level than a company with bespoke customers. Whether this and other factors have to be included in the framework design and whether other factors are not as relevant has to be targeted by future research as well.

6. Validity

In the following, the validity of the multiple-case study conducted in the course of this thesis project is discussed. The four different aspects of construct validity, internal validity, external validity, and reliability are addressed separately. The four aspects are understood as described by Runeson et al in [53].

6.1. Construct validity

The interview questions for the semi-structured interviews were designed by the author based on his understanding of the area. The author furthermore tried to state the questions as clear as possible and explain them furthermore during the interview if necessary. The questionnaire was used in a pilot interview conducted with a researcher familiar with the research topic and research area. After this pilot interview, the questions were discussed with the researcher who has empirical experience in performing several interview based case studies in similar contexts. The questionnaire was refined based on this discussion. However, it is still possible that questions were interpreted differently by different interviewees. This is a potential threat to the construct validity of the research. In order to limit this threat, multiple subjects were interviewed at Company C, D, E, and F. This leads to data triangulation at the named companies and therefore reduces possibly biased information.

6.2. Internal validity

During the case study, a chain of evidence was used to link the coded interviews to the data analysis. Furthermore, the interviews were conducted in a semi-structured way in order to be able to answer follow-up questions where needed or to adapt questions to the specific situation. Additionally, the questionnaire was discussed after the pilot interview in order to add questions that might be relevant to the research questions. A case study report was written for every company based on the data analysis. This case study report was read by the contact persons at each company and possibly also by the interviewees. Feedback was obtained regarding the data analysis. In some cases, this lead to additional discussions regarding some parts of the analysis. However, a thread to internal validity can not be ruled out. As literature directly related to the investigated area is scarce, influencing factors are not yet completely understood.

6.3. External validity

During the multiple-case study, a broad range of different companies was covered. Therefore, the presented state-of-practice covers a broad range as well. The different cases were selected for revelatory reasons and not to ensure triangulation. As the PASTA framework forms the synthesis of this state-of-practice, it should be applicable to a wide spectrum of industry projects as well. However, it will be necessary to validate it in similar cases in order to confirm external validity. This is out of the range of this project and should be tackled in future projects.
As this study was conducted in order to develop an initial framework, further studies with the PASTA framework will be necessary to investigate generalisation possibilities. This is out of scope of this thesis project. The limitations are discussed in detail in chapter 7.

6.4. Reliability

In order to ensure reliability of the study, a number of steps were taken. Firstly, a theoretical framework was developed. This theoretical framework ensures a chain of evidence throughout the whole data collection and data analysis phases. Furthermore, for each case under study, a case study protocol was maintained and a case study report was written in order to avoid deviations from the theoretical framework. The case study reports were read by at least one subject at every company. These subjects then provided feedback on the logical conclusions within the data analysis. Due to time constraints, transcribing and coding of the interviews was done by the author only. This poses a thread to reliability.
The interview questions were reviewed by the researcher involved in the pilot interview. However, it can not be ruled out that some of the questions might have been unclear to interviewees.

7. Conclusions

In the following sections, the thesis work is summarised. Firstly, a summary of the overall work is presented. Secondly, the contributions of the thesis project and its limitations are outlined. Finally, possibilities for future research related to this thesis project are stated.

7.1. Summary

The thesis topic was introduced and outlined in the first chapter 1. A short summary of the area of research was given. The aim of the project was to develop a framework on how to structure GUI acceptance and system tests. In order to reach this aim, four research questions were answered. These were as follows.

- RQ1: How is GUI acceptance and system testing done in practice?
- RQ2: Are there best practices or methodologies in other areas of software engineering which can be directly transferred to GUI acceptance and system testing?
- RQ3: Can these practices or methodologies be merged into a single framework for GUI acceptance and system testing?
- RQ4: Can the resulting framework facilitate GUI acceptance and system test creation, lead to a higher grade of automation, and reduce the maintenance effort of the test suites?

Related work on relevant areas of research was presented in chapter 2.
The first research question was answered through a multiple-case study in multiple teams at six different software-developing companies. The answer can be summarised as follows. GUI system testing plays an important role in industry. In all cases under study, a form of GUI system testing is performed. The practices reach from manual exploratory system testing to automated system testing. The test suite sizes vary greatly between the studied cases. One company had as few as 10 to 15 system tests for a typical product, while another company had over 4000 test cases on system level. Additionally, one of the companies did not document their system tests and performed them in a undocumented, exploratory way. GUI acceptance testing is done less than system teseting and often in an unplanned way. Only two of the studied companies had planned acceptance tests including customer involvement. Two companies had beta tests from time to time. Finally, one company did informal manual acceptance tests where customers navigate through the application. The need for more acceptance testing was seen by many interviewees, but

customer involvement is seen as one of the main factors preventing this. Automation starts to play an important role in GUI acceptance and system testing. All but one case had automated GUI acceptance or system test suites or were working on introducing them. However, a main hindrance here is insufficient tool support. Existing tools are seen to be unsuited or problematic with regard to maintenance costs. Essentially, two different levels of GUI acceptance and system testing were found. Firstly, some companies create large test suites with which they validate many or all of their requirements with test cases. Secondly, small-sized test suites are common which only test the most important scenarios. Especially in agile projects, this test suite level was common.
The second research question can not be answered as clearly. Some overall concepts were found in GUI acceptance and system testing which are also used in software development. This is for instance the use of abstraction concepts for automated GUI acceptance and system testing which are common in software development. However, this is not surprising as automated tests are often written in the same programming languages as the software product. Additionally, the used software process has a strong influence on the GUI acceptance and system testing practices. The concept of only testing the most important scenarios with system or acceptance tests is related to the agile practice of reducing waste. The concept of sharing common episodes or tasks among several scenarios was proposed for scenario management in general in [8]. A similar practice was found in one of the companies and seems to be useful at least for test scenario documentation. For test automation, it might prove valuable as well. It is expected that more relevant practices are found once the proposed framework is evaluated in a real-life project. This is due to the fact that many tasks and approaches are described in this thesis project on a high level. Once they are applied, more relevant research from other areas in software engineering will be valuable as well.
The information acquired in the multiple-case study and from research was merged into the PASTA framework presented in 4.4. The framework is therefore representing the answer to research question three. The PASTA framework consists of two different levels on which GUI acceptance and system testing can be conducted. These two levels represent the two different kinds of GUI acceptance and system testing which were found in the multiple-case study and which seem to work well according to the interviewees. The first level targets base-level GUI acceptance and system testing. Here, only the most important test scenarios are covered and little overhead for test creation, execution, and maintenance is needed. The second level targets organised-level GUI acceptance and system testing. This level requires substantial abstraction and structuring mechanisms in order to be beneficial. It is indicated for long-running projects with high quality requirements, such as safety-critical projects. For each level, project factors are discussed which indicate the use of the respective test suite level or which can be considered for introducing a new GUI acceptance and system test suite in a project. Afterwards, guidelines for implementing both levels are given. They focus on the four areas of test suite design, test scenario elicitation, test automation, and test maintenance. Finally, possible transitions between the two levels are discussed.
The PASTA framework was statically evaluated with the participating companies. The evaluation results were presented in chapter 5. The most beneficial test suite level for each company or project was assessed based on the framework. Additionally, recommendations were given based on the test suite level and the current practices at each respective company or project. These were discussed together with the overall framework in a meeting with the companies. This meeting took place in three of the companies. The feedback from the meetings was mostly positive. The main issues brought up during the meetings were the problem of involving the right customers or customers in general into the test scenario elicitation process, the focus of system testing on requirements validation, and the current state of GUI testing tools. The survey yielded differing results. In addition

to the meetings, all interviewees at four companies were handed out a survey covering the interviews, the framework, the recommendations, and the overall case study. The survey was handed in by 13 subjects. Two main issues arise from the survey results. Firstly, the two test suite levels are seen as not sufficient by multiple survey participants. Secondly, multiple participants stated that the factors indicating or leading to each test suite level are not the main important ones. Both issues should be addressed in future research. Therefore, the fourth research question can not be answered clearly and has to be left for future research.
The validity of the research was discussed in chapter 6.

7.2. Contributions and limitations

The contributions of this thesis to existing research are as follows:
The state-of-practice in GUI acceptance and system testing based on an exploratory multiple-case study at a number of different teams at six companies was assessed. The study focused on identifying common practices in structuring GUI acceptance and system tests, creating these tests, executing the tests, and the usage of tools in the testing context. Additionally, typical problems and solutions in these areas were investigated and presented. The cases under study are teams at software developing companies of varying background. The studied companies are of different size, have differing experience in their area, focus on different markets or sectors, use different software development processes, and employ different testing techniques and approaches.
Based on the gathered information and on current research, a framework on how to approach GUI acceptance and system tests was introduced. The framework consists of two different levels on which GUI acceptance and system testing can be done. Factors for identifying the right level for introducing GUI acceptance and system testing or to determine which level fits the current practice best are provided. Furthermore, guidelines on how to do scenario elicitation, test suite design, test automation, and test maintenance with respect to GUI acceptance and system testing are provided for both levels.
Finally, the framework was evaluated statically with the participating teams in order to identify weaknesses or generally asses the framework.

There are a number on limitations in this thesis project.
Firstly, the number of studied projects for information gathering was limited due to time constraints. Therefore, the gathered information can only pose a small set of practices and experiences in GUI acceptance and system testing. Also, the projects were selected by means of availability and personal contacts. Selection bias can therefore not be ruled out. Furthermore, the means of data collection were restricted to interviews and group sessions in the studied teams. Only qualitative data analysis was performed due to time limitations.
Secondly, the developed framework is not intended to be complete nor is it intended to formalise the test creation or execution process. The author believes that at the current state of GUI testing in practice and research, too much formalisation would hinder the adaptation of any framework.
Thirdly, automation of GUI test creation or execution is not a primary aim of this thesis. The framework presented in this thesis is intended as a starting point which could lead to a higher degree of automation in the future. However, more research will be needed in order to reach this aim.
Finally, again due to time constraints, the evaluation of the framework could only be done statically in the form of discussions with the interviewees and a survey filled out by the interviewees. Therefore, the framework might not be generalisable to every project context.

7.3. Future work

The presented thesis project is intended as a starting point in the area of GUI acceptance and system testing. The case study has shown that there is lots of room for improvements in GUI acceptance and system testing with respect to processes and tools. The interviewees expressed lack of tool support, lack of time, and other problems with their current testing practices. Existing tools require high maintenance effort, are hard to use, or are specific to programming languages or platforms. Manual testing, on the other side, is time-consuming and tedious. While some of the studied companies employ manual acceptance or system testing through the GUI successfully, many interviewees expressed the wish or the need to automate their testing. Concerning processes, testing in general and GUI acceptance and system testing were not properly implemented in many of the studied companies. This was attributed mostly to lack of time and lack of tool support. The PASTA framework could aid in improving the GUI acceptance and system testing practices in software developing projects and increase the benefits from it. However, more experience is needed with the presented framework. The case study focused on studying teams at software developing companies with a broad range of different circumstances. Additionally,the framework could only be evaluated statically. More companies with similar circumstances need to be studied in order to explore if the framework is applicable and generalisable. Additionally, the projects should be studied over a longer period of time in order to evaluate all aspects of the proposed framework. Finally, only qualitative data was collected during the study. By studying additional projects with similar circumstances, quantitative data could be collected in order to verify the findings. All these aspects will be needed in order to validate the framework.

During the static evaluation, a number of issues arose. There are three main issues brought up during the meetings. Firstly, it was stated repeatedly that it is difficult to involve the right customers or to involve customers in general into the testing process. Secondly, not all interviewees agreed on the suggestion that GUI acceptance and system testing should focus mainly on requirements validation and not on bug finding. Thirdly, current GUI testing tools were seen as problematic. Not all interviewees were convinved by the idea of visual GUI testing. These problems apply not only to the proposed framework, but are important to the overall area of GUI acceptance and system testing. Therefore, they could be addressed by future research in the research community in general.

The survey also yielded differing results. Here, two main issues can be seen. Firstly, the subjects participating in the evaluation were unsure whether the two test suite levels are sufficient or not. It might be that more levels have to be introduced in order to apply the framework in real-life projects. This has to be studied in future projects. Secondly, the participants indicated that other factors as the presented ones are more important for choosing the right test suite level. However, they did not state any specific factors. Future projects which try to apply the PASTA framework should target this as well.

The fact that time and cost of testing was commonly mentioned during the interviews also shows the need for cost estimation with respect to the proposed framework. This is out of scope of this thesis but should be a goal for further projects in this area.

Bibliography

[1] Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)* (15 2010), 1 –418.

[2] Fitnesse. `http://fitnesse.org/`, Sept. 2012.

[3] Selenium - web browser automation. `http://seleniumhq.org/`, Sept. 2012.

[4] Skype. `http://www.skype.com/intl/en/home`, Oct. 2012.

[5] Swt: The standard widget toolkit. `http://www.eclipse.org/swt/`, Sept. 2012.

[6] Test management tool = testlink. `http://www.teamst.org/`, Nov. 2012.

[7] Jellytools. `http://wiki.netbeans.org/JellyTools`, Jan. 2013.

[8] Alspaugh, T., Anton, A., Barnes, T., and Mott, B. An integrated scenario management strategy. In *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on* (1999), pp. 142 –149.

[9] Amyot, D., Roy, J.-F., and Weiss, M. Ucm-driven testing of web applications. In *SDL 2005: Model Driven*, A. Prinz, R. Reed, and J. Reed, Eds., vol. 3530 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 1213–1229.

[10] Beck, K., and Andres, C. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[11] Becks, A., and Koller, J. Automatically structuring textual requirement scenarios. In *Automated Software Engineering, 1999. 14th IEEE International Conference on.* (oct 1999), pp. 271 –274.

[12] Berner, S., Weber, R., and Keller, R. Observations and lessons learned from automated testing. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on* (may 2005), pp. 571 – 579.

[13] Börjesson, E., and Feldt, R. Structuring software engineering case studies to cover multiple perspectives. In *SEKE* (2011), pp. 276–281.

[14] Briand, L., and Labiche, Y. A uml-based approach to system testing. *Software and Systems Modeling 1* (2002), 10–42.

[15] Brooks, P. A., and Memon, A. M. Automated gui testing guided by usage profiles. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (2007), ASE '07, pp. 333–342.

[16] Börjesson, E., and Feldt, R. Automated system testing using visual gui testing tools: A comparative study in industry. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (Washington, DC, USA, 2012), ICST '12, IEEE Computer Society, pp. 350–359.

[17] Chang, T.-H., Yeh, T., and Miller, R. C. Gui testing using computer vision. In *Proceedings of the 28th international conference on Human factors in computing systems* (New York, NY, USA, 2010), CHI '10, ACM, pp. 1535–1544.

[18] Cockburn, A. *Agile software development : the cooperative game*, 2. ed. ed. Agile software development series. Addison-Wesley, Upper Saddle River, NJ [u.a.], 2007.

[19] Collins, E., and de Lucena, V. Software test automation practices in agile development environment: An industry experience report. In *Automation of Software Test (AST), 2012 7th International Workshop on* (june 2012), pp. 57 –63.

[20] Craig, Rick D. ; Jaskiel, S. P. *Systematic software testing*, 1. print. ed. Artech House, Boston, Mass, 2002.

[21] Cunning, S., and Rozenbiit, J. Test scenario generation from a structured requirements specification. In *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS '99. IEEE Conference and Workshop on* (mar 1999), pp. 166 –172.

[22] Elbaum, S., Malishevsky, A., and Rothermel, G. Test case prioritization: a family of empirical studies. *Software Engineering, IEEE Transactions on 28*, 2 (feb 2002).

[23] Ganov, S., Khurshid, S., and Perry, D. Symbolic execution for gui testing. In *Proceedings of Testing: Academic and Industrial Conference, Practice and Research Techniques (TAIC PART)* (2007).

[24] Ganov, S., Killmar, C., Khurshid, S., and Perry, D. Event listener analysis and symbolic execution for testing gui applications. In *Formal Methods and Software Engineering*, K. Breitman and A. Cavalcanti, Eds., vol. 5885 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 69–87.

[25] Ghauri, Pervez N. ; Grønhaug, K. *Research methods in business studies*, 4. ed. Financial Times/Prentice Hall, Harlow, 2010.

[26] Grilo, A., Paiva, A., and Faria, J. Reverse engineering of gui models for testing. In *Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on* (june 2010), pp. 1 –6.

[27] Harrold, M. J. Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ICSE '00, ACM, pp. 61–72.

[28] Haugset, B., and Hanssen, G. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE '08. Conference* (aug. 2008), pp. 27 –38.

[29] Heinecke, A., Brückmann, T., Griebe, T., and Gruhn, V. Generating test plans for acceptance tests from uml activity diagrams. In *Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (2010), ECBS '10, pp. 57–66.

[30] Hoda, R., Noble, J., and Marshall, S. The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology 53*, 5 (2011), 521 – 534.

[31] Höfer, A., and Tichy, W. Status of empirical research in software engineering. In *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, V. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, and R. Selby, Eds., vol. 4336 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 10–19.

[32] Itkonen, J., Mantyla, M., and Lassenius, C. How do testers do it? an exploratory study on manual testing practices. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on* (oct. 2009), pp. 494 –497.

[33] Jarke, M., Bui, X. T., and Carroll, J. M. Scenario management: An interdisciplinary approach. *Requirements Engineering 3* (1998), 155–173.

[34] Kundu, D., Sarma, M., Samanta, D., and Mall, R. System testing for object-oriented systems with test case prioritization. *Software Testing, Verification and Reliability 19*, 4 (2009), 297–333.

[35] Li, P., Huynh, T., Reformat, M., and Miller, J. A practical approach to testing gui systems. *Empirical Software Engineering 12* (2007), 331–357.

[36] Linden, F. J. v. d., Schmid, K., and Rommes, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[37] Lions, J. L. Ariane 5 flight 501 failure. Tech. rep., ESA: Ariane 501 Inquiry Board, 1996.

[38] Lowell, C., and Stell-Smith, J. Successful automation of gui driven acceptance testing. In *Extreme Programming and Agile Processes in Software Engineering*, M. Marchesi and G. Succi, Eds., vol. 2675 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 1011–1012.

[39] Marchetti, E., Schilders, L., and winfield, s. Scenario-based testing applied in two real contexts: Healthcare and employability. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on* (march 2011), pp. 89 –98.

[40] Martin, D., Rooksby, J., Rouncefield, M., and Sommerville, I. 'good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on* (may 2007).

[41] Melnik, G. I. *Empirical Analysis of Executeable Acceptance Test Driven Development*. PhD thesis, 2007.

[42] Memon, A. M. GUI testing: Pitfalls and process. *Computer 35*, 8 (2002), 87–88.

[43] Memon, A. M. Using tasks to automate regression testing of GUIs. In *Proceedings of The IASTED International Conference on ARTIFICIAL INTELLIGENCE AND APPLICATIONS (AIA 2004)* (Innsbruck, Austria, Feb. 2004).

[44] Memon, A. M., and Xie, Q. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. *IEEE Trans. Softw. Eng. 31*, 10 (2005), 884–896.

[45] Mu, B., Zhan, M., and Hu, L. Design and implementation of gui automated testing framework based on xml. In *Software Engineering, 2009. WCSE '09. WRI World Congress on* (may 2009), vol. 4, pp. 194 –199.

[46] Mugridge, R. *Fit for developing software : framework for integrated tests*. Prentice Hall PTR, Upper Saddle River, N.J., 2005.

[47] Nebut, C., Fleurey, F., Le Traon, Y., and Jezequel, J.-M. Requirements by contracts allow automated system testing. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on* (nov. 2003), pp. 85 – 96.

[48] Nebut, C., Fleurey, F., Le Traon, Y., and Jezequel, J.-M. Automatic test generation: a use case driven approach. *Software Engineering, IEEE Transactions on 32*, 3 (march 2006), 140 – 155.

[49] North, D. Introducing bdd. *"Better Software Magazine* (march 2006). Available at *http://dannorth.net/introducing-bdd/*, Accessed December 12, 2012.

[50] Paiva, A. C., Faria, J. C., and Vidal, R. F. Towards the integration of visual and formal models for gui testing. *Electronic Notes in Theoretical Computer Science 190*, 2 (2007), 99 – 111.

[51] Pohl, K., Böckle, G., and Linden, F. J. v. d. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[52] Rogers, R. Acceptance testing vs. unit testing: A developer's perspective. In *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, C. Zannier, H. Erdogmus, and L. Lindstrom, Eds., vol. 3134 of *Lecture Notes in Computer Science.* Springer Berlin / Heidelberg, 2004, pp. 244–255.

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

[54] Ryser, J., and Glinz, M. A scenario-based approach to validating and testing software systems using statecharts. In *In 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99* (1999), p. 7.

[55] Sjøberg, D., Dyba, T., and Jorgensen, M. The future of empirical methods in software engineering research. In *Future of Software Engineering, 2007. FOSE '07* (may 2007).

[56] Solis, C., and Wang, X. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (2011), pp. 383 –387.

[57] Somé, S. S., and Cheng, X. An approach for supporting system-level test scenarios generation from textual use cases. In *Proceedings of the 2008 ACM symposium on Applied computing* (New York, NY, USA, 2008), SAC '08, ACM, pp. 724–729.

[58] Srikanth, H., and Banerjee, S. Improving test efficiency through system test prioritization. *Journal of Systems and Software 85*, 5 (2012), 1176 – 1187.

[59] Tinkham, A., and Kaner, C. Exploring Exploratory Testing. In *STAR East conference* (2003).

[60] Tsai, W., Saimi, A., Yu, L., and Paul, R. Scenario-based object-oriented testing framework. In *Quality Software, 2003. Proceedings. Third International Conference on* (nov. 2003), pp. 410 – 417.

[61] White, L., and Almezen, H. Generating test cases for gui responsibilities using complete interaction sequences. In *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on* (2000).

[62] Xie, Q., and Memon, A. M. Studying the characteristics of a 'good' GUI test suite. In *Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006)* (Nov. 2006), IEEE Computer Society Press.

[63] Yin, R. K. *Case study research : design and methods*, 4. ed. Applied social research methods series ; 5. Sage, Los Angeles, CA, 2009.

Appendix

A. Questionnaire

The following is a sample questionnaire used in the case study of *Company A* and *Company B*. For other cases under study, depending on the case context, some questions were removed, changed or added. For instance, the first questions were already answered in a focus group in the study at *Company C*. Therefore, they were left out.
Questions which likely need a long answer and therefore take more time are underlined. This was done as a guidance for the author.

1. Could you shortly tell me about the company you are working with?
2. How big is the company we are talking about?
3. What is the product you are working on, What is it called?
4. What is your position and role in the project?
5. How long have you been working with the project?
6. How would you rate your understanding of the product?
7. How established is the company in the area your project is in?
8. Who are the customers of your product?
9. What platforms does the product run on?
10. How old is the product?
11. What is the size of the product?
12. Can you describe the process that is being used in your team?
13. Is there a difference between a tester and a developer? Do the roles change over time?
14. How flexible is the process?
15. How big is the whole team working on the product?
16. What is an acceptance test for you?
17. What is a system test for you?
18. Are GUI acceptance and/or system tests used in your project?
19. Which tools are used for GUI testing?
20. Which tools have been used for GUI testing, but have been abandoned again? Why?
21. Is the GUI testing tool prescribed by boss/customer/manager/etc.?
22. How big is the test suite for GUI system and acceptance tests?

23. Are customers involved in the acceptance testing process?
24. Do customers want to be involved?
25. Where do the scenarios for the test cases come from? Who comes up with them?
26. How are the test cases created?
27. Are artefacts from earlier project phases reused?
28. Are certain artefacts from previous phases/iterations which are reused in the test design?
29. Can artefacts from earlier parts of the project be reused without substantial modifications?
30. Are there any scenarios which can not be tested using the current tools?
31. Are there multiple test cases which share common artefacts? That is, are there test cases which are connected?
32. How is the connection between test cases modelled?
33. How does the maintenance of the test suite look like?
34. How fast does the GUI evolve?
35. Does the evolution usually go in steps only (e.g. releases)?
36. Are any parts of the product developed by other teams or even other companies?
37. What are the most important product attributes to customers/management/bosses?
38. Where do you see problems with the current process/way of working?
39. Do you see major drawbacks with your team's current testing practices?
40. How could possible improvements look like?
41. How would you rate your current testing practices?

B. Evaluation survey

For evaluation purposes, a survey was conducted among all subjects interviewed in the course of the multiple-case study. The survey was created and hosted on a free survey service on the internet. As a first question, the employing company was asked of every participant. This was done to sort the results by company. Afterwards, the survey was divided into three different categories regarding the developed framework, the recommendations made for every company, and the overall case study. The content of each category is explained in the following. Please note that in the original survey and during the case study, the framework was instead referred to as a *methodology*. In the following, the word methodology from the original survey was therefore replaced with the word framework.

B.1. Framework

In the framework category, the survey participants were asked to rate their agreement for seven different statements concerning the developed framework. They could rate them on a scale from *Strongly agree* to *Strongly disagree* with five different options. Furthermore, comments could be provided for every statement. The statements were as follows:

1. The framework is structured in a logical way.
2. The two different testing levels are suitable to describe most real-world projects.
3. The factors leading to both levels are relevant for the decision.
4. The framework presents solutions or ideas relevant for the project(s) I am involved in.
5. I can see the connection between the framework and the interview questions.
6. I would have expected a different focus given the interviews.
7. I think the framework can be applied in future software projects.

Finally, a free text field was provided for leaving general comments regarding the framework category.

B.2. Recommendations

Similarly to the framework category, five different statements were provided in the recommendations category. The same rating scale was applied as in the previous category. Additionally to the rating, comments could be provided for every statement. The statements were as follows:

1. The recommendations are valuable for our project(s).
2. Some recommendations do not fit to our specific situation.
3. I would have placed my company in another test suite level based on the developed framework.
4. Other factors are more relevant for placing a project in one of the two testing levels.
5. I will try to implement some of the recommended things.

Finally, a free text field was provided for leaving general comments regarding the recommendations category.

B.3. Overall

Two more statements were made in the last category. These concerned the overall case study. The same scale was used as in the other two categories. Additionally to the rating, comments could be provided for every statement. The statements were as follows:

1. The cooperation was valuable for the project.
2. The author seemed to be competent in the investigated area.

After the statements, the survey participants were asked to give an overall rating to the case study. The scale reached from one star to five stars with one star being the lowest score and five stars the highest.
Finally, a free text field was provided for leaving general comments regarding the case study.

C. Survey results

In the following, the survey results are listed by company. Additionally, the case study ratings are listed.

C.1. Company C

Question	Answers
The framework is structured in a logical way.	Strongly agree, Agree, Agree, Agree
The two different testing levels are suitable to describe most real-world projects.	Neutral, Neutral, Agree, Strongly agree
The factors leading to both levels are relevant for the decision.	Agree, Agree, Agree, Agree
The framework presents solutions or ideas relevant for the project(s) I am involved in.	Agree, Neutral, Neutral, Agree
I can see the connection between the framework and the interview questions.	Agree, Agree, Neutral, Strongly agree
I would have expected a different focus given the interviews.	Disagree, Neutral, Agree, Agree
I think the framework can be applied in future software projects.	Disagree, Neutral, Neutral, Strongly agree
The recommendations are valuable for our project(s).	Neutral, Neutral, Neutral, Agree
Some recommendations do not fit to our specific situation.	Agree, Agree, Agree, Neutral
I would have placed my company in another test suite level based on the developed framework.	Agree, Disagree, Neutral, Neutral
Other factors are more relevant for placing a project in one of the two testing levels.	Agree, Agree, Agree, Agree
I will try to implement some of the recommended things.	Agree, Agree, Agree, Agree
The cooperation was valuable for the project.	Agree, Neutral, Neutral, Strongly agree
The author seemed to be competent in the investigated area.	Agree, Agree, Agree, Strongly agree

Table C.1.: Survey results for Company C

As an overall case study rating (one to five; the higher the better) the survey participants filled in three, three, three, and five.

C.2. Company D

Question	Answers
The framework is structured in a logical way.	Agree, Agree, Neutral
The two different testing levels are suitable to describe most real-world projects.	Neutral, Neutral, Agree
The factors leading to both levels are relevant for the decision.	Agree, Neutral, Agree
The framework presents solutions or ideas relevant for the project(s) I am involved in.	Agree, Agree, Neutral
I can see the connection between the framework and the interview questions.	Strongly agree, Agree, Agree
I would have expected a different focus given the interviews.	Disagree, Agree, Disagree
I think the framework can be applied in future software projects.	Neutral, Neutral, Agree
The recommendations are valuable for our project(s).	Neutral, Agree, Neutral
Some recommendations do not fit to our specific situation.	Neutral, Agree, Agree
I would have placed my company in another test suite level based on the developed framework.	Disagree, Disagree, Disagree
Other factors are more relevant for placing a project in one of the two testing levels.	Agree, Agree, Disagree
I will try to implement some of the recommended things.	Neutral, Neutral, Neutral
The cooperation was valuable for the project.	Agree, Agree, Agree
The author seemed to be competent in the investigated area.	Agree, Agree, Agree

Table C.2.: Survey results for Company D

As an overall case study rating (one to five; the higher the better) the survey participants filled in three, four, and four.

C.3. Company E

Question	Answers
The framework is structured in a logical way.	Strongly agree, Agree, Agree, Agree
The two different testing levels are suitable to describe most real-world projects.	Agree, Neutral, Neutral, Agree
The factors leading to both levels are relevant for the decision.	Agree, Strongly disagree, Neutral, Neutral
The framework presents solutions or ideas relevant for the project(s) I am involved in.	Strongly agree, Strongly disagree, Disagree, Disagree
I can see the connection between the framework and the interview questions.	Strongly agree, Agree, Agree, Agree
I would have expected a different focus given the interviews.	Disagree, Agree, Neutral, Neutral
I think the framework can be applied in future software projects.	Agree, Agree, Neutral, Disagree
The recommendations are valuable for our project(s).	Agree, Disagree, Disagree, Neutral
Some recommendations do not fit to our specific situation.	Strongly disagree, Neutral, Neutral, Agree
I would have placed my company in another test suite level based on the developed framework.	Strongly disagree, Disagree, Disagree, Disagree
Other factors are more relevant for placing a project in one of the two testing levels.	Disagree, Neutral, Neutral, Neutral
I will try to implement some of the recommended things.	Agree, Neutral, Disagree, Disagree
The cooperation was valuable for the project.	Neutral, Neutral, Neutral, Neutral
The author seemed to be competent in the investigated area.	Agree, Neutral, Agree, Agree

Table C.3.: Survey results for Company E

As an overall case study rating (one to five; the higher the better) the survey participants filled in empty, two, three, and three.

C.4. Company F

Question	Answers
The framework is structured in a logical way.	Strongly agree, Agree
The two different testing levels are suitable to describe most real-world projects.	Agree, Agree
The factors leading to both levels are relevant for the decision.	Agree, Agree
The framework presents solutions or ideas relevant for the project(s) I am involved in.	Agree, Neutral
I can see the connection between the framework and the interview questions.	Strongly agree, Neutral
I would have expected a different focus given the interviews.	Disagree, Disagree
I think the framework can be applied in future software projects.	Neutral, Agree
The recommendations are valuable for our project(s).	Agree, Agree
Some recommendations do not fit to our specific situation.	Agree, Neutral
I would have placed my company in another test suite level based on the developed framework.	Strongly disagree, Disagree
Other factors are more relevant for placing a project in one of the two testing levels.	Neutral, Neutral
I will try to implement some of the recommended things.	Agree, Agree
The cooperation was valuable for the project.	Agree, Agree
The author seemed to be competent in the investigated area.	Strongly agree, Agree

Table C.4.: Survey results for Company F

As an overall case study rating (one to five; the higher the better) the survey participants filled in four and four