2013 39th Euromicro Conference Series on Software Engineering and Advanced Applications

State-of-Practice in GUI-based System and Acceptance Testing: An Industrial Multiple-Case Study

Grischa Liebel, Emil Alégroth, Robert Feldt
Software Engineering and Technology
Chalmers University of Technology
Göteborg, Sweden
liebel@student.chalmers.se, {emil.alegroth, robert.feldt}@chalmers.se

***Abstract*—Software testing is an essential means of evaluating software quality. System and acceptance tests aim to validate a system's conformance to its requirements on a high level of system abstraction. Therefore, they are generally performed by executing end-user scenarios through the system's graphical user interface (GUI). However, to the authors' best knowledge, there are no empirical studies that evaluate how GUI-based system and acceptance testing is performed in industrial practice. In this paper, we present a multiple-case study with the goal to investigate the state-of-practice of GUI-based system and acceptance testing at six software development companies of varying context. The main findings are that manual, GUI-based system testing is widespread and that automated GUI-based system and acceptance testing exists only on a small scale. Additionally, the study identifies core problems with GUI-based system and acceptance testing such as test tool limitations, high test costs and customer involvement in testing.**

***Keywords*—Software Testing; Acceptance Testing; System Testing; GUI-based Testing; Empirical Study; Case Study; Quality assurance; Process improvement**

I. Introduction

Software testing is a widely-accepted practice to identify faults in a software system during its development. Software can be tested on different levels of system abstraction [10]. Commonly, these levels are categorised as unit, integration, system, and acceptance tests, where unit tests are performed on a lower level of system abstraction, e.g. on system components, and acceptance tests on a higher level of abstraction, e.g. the graphical user interface (GUI). The focus of this paper is on *system* and *acceptance* tests which aim to validate if the system fulfills its specifications and requirements. That is, tests that are performed on the running system after all parts of the system are integrated. Thus, in this paper we define system testing as *"Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements."* [1]. Whilst acceptance testing is defined as *"Formal testing conducted to enable a user, customer, or other authorised entity to determine whether to accept a system or component."* [1]. The two testing approaches differ mostly with regard to the input data and the stakeholder conducting the tests. That is, system tests are typically conducted by developers or dedicated testers whilst acceptance tests are typically conducted by customers.

Software testing is costly and studies have shown that as much as fifty percent of the overall software development cost can be related to testing [13]. By automating manual tests, these costs could be reduced. However, automated tests can not replace manual tests completely [6]. One reason is that scripted test scenarios can not find faults outside said scenario. Thus, either more scripts have to be developed for an abundance of scenarios or the tests are complemented with manual exploratory testing. The latter alternative is more common in industrial practice due to lower cost. Additionally, many existing approaches for automated system testing suffer from limitations [8]. For instance, tests written with so-called Record&Replay tools are often fragile to GUI changes, making them costly to maintain. Instead, a common industrial practice is to run subsets of manual test cases chosen through test case prioritisation [11]. By selecting only the most important tests and executing those, the testing costs are perceivably reduced. However, this technique lowers test coverage and can therefore have detrimental effects on system quality. In addition, the selection process is complex, especially for large systems. Hence, different system test approaches exhibit unique and/or common limitations in terms of fragility, complexity and high cost. Thus, there is no single best practice how to perform system and acceptance testing.

As graphical user interfaces (GUIs) have become ubiquitous in most software systems, more focus has been placed on GUI-level testing, i.e. testing of the system's functionality through the GUI. However, even though GUI testing has many familiarities with traditional software testing, it needs to be approached differently [20]. For instance, it is stated that traditional test coverage criteria are not useful for GUI testing [20]. As GUI components and the underlying code have different abstraction levels, it is often not obvious which parts of the code are stimulated by which GUI events. Hence, new challenges, such as defining appropriate test criteria, arise when testing systems through their graphical user interfaces. In this paper we define *test coverage* of a software system as the *"extent to which the test cases test the requirements for the system or software product."* [1].

Thus, system and acceptance testing remain problematic. Additionally, to the authors' best knowledge, there is a lack of empirical studies on the state-of-practice in GUI-based system and acceptance testing. Consequently, there is a gap in body of knowledge and a need for empirical research regarding the current state of practice of, and the industrial practitioners'

978-0-7695-5091-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SEAA.2013.29

challenges with, GUI-based system and acceptance testing.

This paper presents the results of an empirical study conducted in multiple teams at six different software developing companies. The study was exploratory and aimed at investigating the current state-of-practice in system and acceptance testing in industry. Data acquisition was performed through semi-structured interviews with one or multiple interviewees at each company. Results of the study show that manual GUI-based system and acceptance testing is common in industrial practice. However, automation for these tests is both wanted and needed in order to lower testing costs and reduce the time invested in testing. Common challenges that were identified with GUI-based system and acceptance testing are limitations in the testing tools, cost of the testing, handling of test data, lacking customer involvement, and testability of legacy systems. Hence, many of the challenges are common to general automated testing. These results help make up the papers main contributions, which are:

1) An account of the current state of practice of GUI-based system and acceptance testing in industry.
2) Common problems and limitations in the area of GUI-based system and acceptance testing out of the practitioners' view.

These contributions can help to guide future research in the area of GUI-based system and acceptance testing and improve the testing processes in industrial practice.

The rest of this paper is structured as follows. Related work on the area of GUI-based system and acceptance testing is discussed in section II. The research methodology underlying the empirical study is discussed in section III together with the study's validity. Section IV covers the results of the empirical study and their analysis. The paper is concluded in section V.

II. Related Work

The study presented in this paper aims at identifying the state-of-practice of system and acceptance testing performed through a system's GUI. While there are numerous publications on acceptance testing, e.g. [14], [19], and system testing, e.g. [7], [22], [17], few focus on how to perform system or acceptance tests through the GUI.

Generally, system-level tests can be executed manually or automatically. Manually, a system can be tested through its GUI by releasing beta versions to the product's customers [20]. The customers can then use the beta version and report possible faults to the developers. Additionally, exploratory testing can be used to manually test a system through its user interface [28]. In exploratory testing, the tester creates test cases actively while using the system under test with his or her knowledge of the system or similar systems. Apart from these approaches, research on manual testing is rare [15].

Many approaches in system testing research are focused on the prioritisation of test cases. These techniques aim at detecting the maximum number of faults in a software system by only executing a part of the overall test suite. An empirical study on test prioritisation techniques is presented in Elbaum et al. [11]. It shows that prioritisation techniques can improve the rate of fault detection in comparison to random selection of test cases [11]. Another approach to prioritise system test cases based on requirements is presented in Srikanth and Banerjee [26]. The results indicate that the used prioritisation technique can improve the rate of failure detection compared to a random selection of test cases.

Another common topic in research is to automatically derive system test scenarios using models, i.e. Model-Based Testing. These test scenarios can then be executed manually or automatically using testing tools. In Nebut et al. [22], a modified version of UML use case diagrams is used to create so-called *test objectives*. These test objectives can then be used as test cases for system testing. Briand and Labiche introduce the TOTEM methodology to create system test requirements and test cases by using UML models created for other purposes, e.g. system modelling [7]. Nebut et al derive test cases for embedded, object-oriented software from documented requirements [23]. Hence, models can be used to automatically create test cases and thus speed up the test creation process. Test case creation overhead is avoided if the models were created for different purposes, e.g. for system modelling.

The main focus in acceptance testing research lies on presenting the customer with an easy-to-use tool and a syntax to write test scenarios. A popular tool in this area is FIT [21] and a multitude of specialised tools building on the FIT framework, like FitNesse for testing web applications [2]. These tools are designed to be understandable without expert or programming knowledge. An industrial case study by Haugset and Hannssen presents effects which arise in industry projects using FIT [14]. It is reported that developers that use FIT get a better overview of the project's status and the actual function of the software system. Practical experience with FIT projects is collected and presented in Melnik [19]. It is stated that the main challenge with executable automated acceptance tests is maintaining the tests because of weak tool support. Hence, the main focus of research in acceptance testing is to develop acceptance testing tools which are easy to use. However, maintaining tests remains a challenge.

In the area of automated, GUI-based testing, so-called *Capture and Replay* tools, such as the open source web browser automation tool Selenium IDE [3], are used due to their ease of use. These tools record the interactions with the GUI by identifying GUI components on a GUI-component level, i.e. beneath the bitmap GUI level shown to the user, or by recording mouse coordinates. However, the scripts can break easily if references to the GUI components change or if the GUI layout changes [16]. Therefore, it is often stated that test cases automated with *Capture and Replay* tools require a high maintenance effort. In order to avoid the inherent problems with Capture and Replay tools, Chang et al. present their tool Sikuli in Chang et al. [9]. Sikuli uses computer vision to detect interaction with the GUI instead of recording coordinates or identifying GUI components on code-level. Therefore, the tests are not as sensitive to changes in the graphical user interfaces as tests implemented using Capture and Replay tools. Börjesson and Feldt name this approach *Visual GUI Testing (VGT)* [8]. A study at Saab AB conducted by the authors shows initial support that automated system testing using VGT can be used in industry.

Hence, current research on system-level testing aims at automatically deriving test cases, prioritising test cases, de-

velopment of tools which facilitate acceptance testing for customers, or development of tools which can record and replay user interaction with a system. Automated tests can facilitate the test process and lower costs. However, in manual testing more defects are found as the knowledge and experience of the tester is used during the testing. Therefore, it is often stated that automated testing can never fully replace manual testing [6], [15]. Still, research on manual testing is rare [15]. Existing techniques for automated test scenario creation are costly and often require an extensive upfront cost, e.g. for script development or modelling, as well as expert tool knowledge. It is therefore unclear if these techniques can be used in practice. While initial support for system testing using Visual GUI Testing tools is shown in Börjesson and Feldt [8], the costs of creating and maintaining the test suite are still unknown and are therefore a subject of future work. Therefore, there is more research required on both manual and automatic GUI-based system and acceptance testing. Especially studies which cover the whole testing process and studies into the industrial practice in GUI-based system and acceptance testing that are currently under-represented in research but which could be beneficial to both academics and industrial practitioners [18].

III. Research Methodology

The study presented in this paper was conducted using a multiple-case study design. The cases were selected in a way that "contrasting results but for anticipatable reasons" could be expected [29]. That is, teams within companies of different size and characteristics were chosen in order to acquire a broad view of the GUI-based system and acceptance testing practices. The aim of the study was to determine the current state of practice of GUI-based system and acceptance testing in industry. Two research questions guided the study and outline its focus. The first research question **RQ1**, *"How is GUI-based system and acceptance testing performed in industry?"*, is aimed to determine the general state of GUI-based system and acceptance testing in industry. The second research question **RQ2**, *"Which are the typical problems related to GUI-based system and acceptance testing in industrial practice?"*, focusses on current hurdles and deficits in the area of GUI-based system and acceptance testing.

A. Case Study Design and Data Collection

The study was exploratory in nature, i.e. its aim was to "find what is happening, seek new insights and generate ideas and hypotheses for new research" [25]. As outlined in section II, there are only a few directly related works on the topic of GUI-based system and acceptance testing. Therefore, using an exploratory case study design is justified in order to understand the current state-of-practice.

The case study design is visualised in Figure 1.

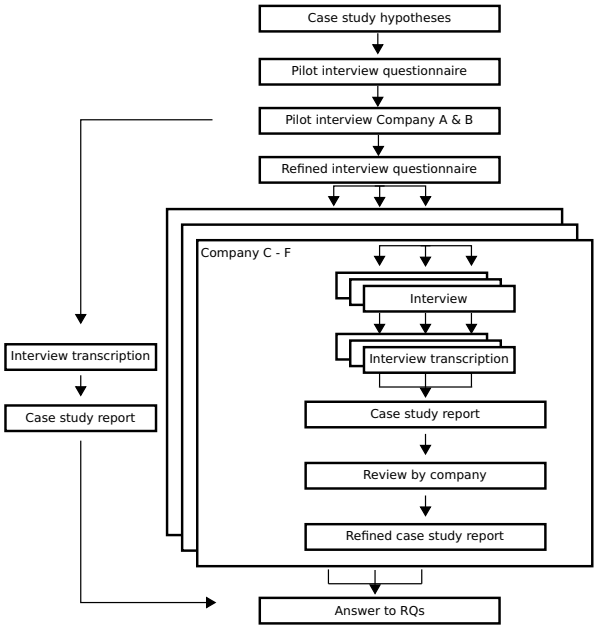

Fig. 1. Research Methodology

The study was based on 9 case study hypotheses which were evaluated in each case. These hypotheses were derived based on the current state of research in the area of GUI-based system and acceptance testing and are as follows.

1) GUI-based 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-based testing tools are FitNesse, FIT, and Selenium.
6) Maintenance of test suites is problematic.
7) There is a clear need for improvement of GUI-based acceptance and system testing practices.
8) GUI testing in general and GUI-based acceptance and system testing in particular does not play an important role in industry.
9) The reasons against a wide adoption of GUI-based 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.

The primary data for the study was collected using twelve semi-structured interviews. According to Ghauri and Grønhaug "Unstructured interviews are advantageous in the context of discovery." [12]. In addition, semi-structured interviews were used because each interview was limited to one hour. The interview questions were derived based on the hypotheses. The companies were selected both based on availability and on their characteristics in order to acquire data from a variety of contexts. However, the first two companies were selected because they expressed interest in the study. Afterwards, further companies with different characteristics were contacted in order to broaden the study. For anonymity reasons the companies are not named in this paper, but are instead referred to as *Company A* to *Company F*. Within the companies, different teams were studied. An overview of the companies with number of employees, business strategies, and the studied

teams is shown in Figure 2. The continuation of this section will describe the participating companies in more detail.

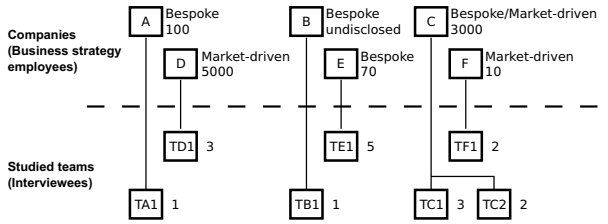

Fig. 2. Cases under study

Company A develops safety-critical air traffic management software. The company uses a toolbox of different development processes depending on the project. There are some projects which are run in a plan-driven way, other projects are developed according to agile processes.

Company B develops military-grade software for bespoke customers. Similarly to Company A, the software is safety-critical. Company B develops software according to a plan-driven process which is close to the waterfall model. In every iteration, they do four to six weeks of manual system testing.

Company C is a global company developing ERP software. Their software is used in mid-size to large-size global companies and is available in multiple languages. The development is done using an adapted version of Scrum.

Company D is a global company offering web hosting services. Most of the software development at Company D is in the form of web applications. In the course of this study, the quality assurance department of the company's web hosting division was studied. The studied team TD1 is involved in both waterfall and agile projects.

Company E develops software and offers consultancy services related to software development. The development is usually following agile processes and projects typically run for a year or less.

Company F is a market-driven start-up company developing software for the security sector. The development is done using agile processes. In the last year, the main goal of Company F was to release new functionality. Now, they are in the process of introducing more quality assurance and testing.

The teams under study were located at different places in Germany, Romania, Sri Lanka, and Sweden, some with distributed development locations. The teams within every company were selected by the contact person at the respective company depending on suitability and available time for conducting the case study. The interview guide used during the interviews with the six companies was structured according to a hierarchical question structure. That is, questions referring to the company or the specific project were asked first, detailed questions about the testing practices were asked later on. A first version of the questionnaire was used for a pilot interview for teams TA1 and TB1. This first interview was conducted with a company proxy. That is, the interviewee was not an employee at any of the companies where TA1 or TB1 resided but had in-depth knowledge of the two companies. The interviewee was involved in a research project with both companies in GUI test automation. Therefore, he had in-depth knowledge of the testing practices at Company A and B. The questionnaire was then refined and used for all interviews in Company C, D, E and F. The interviewees were developers and testers employed at their respective company for six months to 14 years. An overview over the interviewees and their context is given in Table I.

TABLE I. INTERVIEW DETAILS

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

For every studied case, a case study protocol was created and maintained during the study. It consists of a short introduction to the case study, the case study hypotheses, a short description of the studied case, a short outline of the data collection process and timing, and the structure of the final case study report. The case study report was created after the completion of the interviews and the data analysis. It was sent out to the contact persons at each company and was discussed with them. Teams TA1 and TB1 are special cases as they were not accessed directly but through the company proxy, making it secondary data as defined by Ghauri et al. [12]. However, the proxy's understanding of the testing practices at the companies wa deep enough to answer questions related to this research topic.

After each interview, the interview was transcribed and coded. The set of codes was derived during the coding process and not *a priori*. Starting from the coded transcriptions, the case study hypotheses were evaluated for every company separately. Where possible, multiple statements from different interviews were used in order to evaluate a case study hypothesis and ensure validity of the approach.

B. Validity

In the following section, the validity of the multiple-case study presented in this paper is discussed. The four different aspects of construct validity, internal validity, external validity, and reliability are addressed separately [25].

1) Construct validity: The authors have previous experience in conducting interviews, in performing case studies, and in the area of GUI-based system and acceptance testing. Based on this experience, the interview questions for the semi-structured interviews were designed. The authors tried to state the questions unambiguously and explain them furthermore during the interview if necessary. Additionally, the pilot interview at TA1 and TB1 was used to refine the interview guide. This is a potential threat to the construct validity of the research. In order to limit this threat, multiple subjects

were interviewed at TC1, TC2, TD1, TE1, and TF1. This leads to data triangulation at the named teams and therefore reduces possibly biased information. Additionally, a case study report was handed out to the interviewees and the contact person at each company containing a short description of the current state of GUI system and acceptance testing and the evaluation of the case study hypotheses in the studied teams. Where necessary, corrections were made based on the feedback from the study participants. For instance, the authors concluded from the interviews at TF1 that testing was not introduced because of time limitations. However, the interviewees later presented that this was partly related to the lack of testability of their software system. This part had not been discussed in the interviews and was therefore unknown to the authors.

2) Internal validity: Every case study hypothesis was connected to a number of codes. Therefore, every conclusion made during the data analysis is connected to the interviews by a clear chain of evidence. A case study report was written for every company and read by the contact persons at each company and possibly also by the interviewees. Thus, feedback on the data analysis was obtained and wrong conclusions or statements could be identified and corrected. However, a threat 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.

3) External validity: During the multiple-case study, a broad range of different companies was studied. Therefore, the presented state-of-practice covers a broad range of practices in GUI-based system and acceptance testing as well. The different cases were selected for revelatory reasons and not to yield similar results. Therefore, more cases should be studied in order to understand if the results can be generalised to a wider spectrum. This is out of the range of this project and should be tackled in future projects.

4) Reliability: In order to ensure reliability of the study, a number of steps were taken. Firstly, the case study hypotheses were developed. These helped to guide the study at every team. Furthermore, the used case study protocols and case study reports helped to avoid deviations from the study's aim and the case study hypotheses. Due to time constraints, transcribing and coding of the interviews was done by one of the authors only. This poses a threat to reliability. The interview questions were reviewed by the researcher involved in the pilot interview and explained to the interviewees where necessary. However, it can not be ruled out that some of the questions might have been unclear to interviewees.

IV. Results & Analysis

In the following section, the results of the multiple-case study are presented and discussed. Firstly, the two research questions are answered separately. Finally, the findings in general are discussed.

A. RQ1

How is GUI-based system and acceptance testing performed in industry?

The practices observed at the studied teams with regard to GUI-based system and acceptance testing can be summarised as follows:

1) Manual GUI-based system testing is widespread.
2) Formalised GUI-based acceptance testing is not common.
3) Automated GUI-based system and acceptance testing exists mostly on a small scale.

GUI-based system testing plays an important role in industry. In all cases under study, a form of GUI-based system testing was performed. Team TA1 in Company A has roughly 65 manual system test cases in one of their main products. Team TB1 in Company B has 40 system test suites built from from 4000 use cases. Three of these test suites have been automated using Visual GUI Testing. Four to six weeks are spend on system testing each iteration. In Company C, team TC1 performs automated system testing, while team TC2 uses a test suite consisting of 150 test cases for manual system testing. Team TD1 uses manual testing in projects following a waterfall process. In agile projects, the most important system test cases are automated using Selenium WebDriver [3]. In contrast to Selenium IDE, tests for Selenium WebDriver are not simply recorded, but implemented in a programming language. The remaining test cases are executed manually. The number of automated test cases and overall system test cases varies depending on the size and duration of each project. According to one interviewee, a typical test suite size is 50 system test cases for small projects and over 1000 for large projects. Team TE1 uses automated system testing with Selenium WebDriver as well. They have around 15 system test cases in a typical project. The test cases are run automatically on continuous integration servers during every build of the project. Team TF1 does exploratory system testing. Developers manually test functionality written by other developers. The tests are not documented and the time invested in testing varies. The team decided to use this to prioritise development instead of extensive testing. Additionally, it was brought up that the system suffers from testability limitations that makes automated testing difficult.

GUI-based acceptance testing is less common than system testing and often performed in an unplanned way. Teams TA1 and TB1 have planned acceptance tests including customer involvement as they develop safety-critical software. The test cases are brought by the customers to the acceptance tests. Both team TD1 and team TF1 do beta testing. The product is send out to a selected number of customers who then manually test the application and report possible errors. In team TD1, the interviewees stated that this is done "from time to time". In TF1, the interviewees stated that beta testing was done in the past. TE1 does acceptance testing through the GUI with customer involvement at every retrospective meeting. The customer then runs through the application and decides if new features have been implemented successfully or not. However, the test cases are not documented and the whole testing process is not formalised.

Automation starts to play an important role in the area of GUI-based system and acceptance testing. All but one case had automated GUI-based system or acceptance test suites or were working on introducing them. Both TA1 and TB1 have recently introduced the Visual GUI Testing tool Sikuli and automated parts of their system test suites. In both teams, first experiences are promising [8], [5]. Team TC1 at Company C uses the Jelly framework included in NetBeans [4], a GUI testing

tool for GUIs written in Java Swing. The team faces some limitations with this tool, such as problems with identifying GUI components in a new version of Jelly. However, they are generally satisfied with the use of Jelly. Both TD1 and TE1 use Selenium WebDriver for test automation and are satisfied with this choice.

Essentially, two different levels of GUI-based system and acceptance testing were identified during the study. Firstly, some companies create large test suites with which they validate many or all of their requirements with test cases. TA1, TB1, and TD1 fit in this category. Secondly, small-sized test suites which only test the most important system functionality are common. Especially in agile projects, this test suite level was common. This approach was found in TC1 and TE1. TF1 has so far only unformalised, manual testing, but are planning to introduce more testing in the coming months.

B. RQ2

Which are the typical problems related to GUI-based system and acceptance testing in industrial practice?

Numerous problems in the area of GUI-based system and acceptance testing were identified during the study. These can be summarised as follows:

1) Limitations of automated testing tools.
2) High costs of testing in general.
3) Introduction of customer involvement.
4) Handling of test data.
5) Testability of legacy software.

All teams but TF1 face some problems related to automated testing tools. Automated GUI-based system testing using Visual GUI Testing has recently been introduced in TA1 and TB1. The interviewees at TC1 and TC2 all stated problems with the existing automated testing tools. These ranged from high maintenance costs of the resulting test cases to actual tool limitations like missing functionality. For instance, one of the interviewees stated that the current version of Jelly is unable to identify some buttons on the GUI. In TD1, minor limitations with Selenium WebDriver were reported by the interviewees. These are related to complex GUI behaviour like Drag and Drop functionality which is hard to implement in Selenium WebDriver. Additionally, the interviewees stated that they could usually only automate part of their system test suite as otherwise maintenance costs would get too high. However, this was only an estimation of the interviewees as they had not tried to automate a complete test suite, yet. In TE1, no direct problems with the used system testing tool was reported. However, when asked about the acceptance of the tool within team TE1, the interviewees stated that it was accepted as "the lesser evil" in comparison to other testing tools. This shows that there is still room for improvement with respect to automated testing tools at TE1. TF1 has not yet introduced any formalised testing. Additionally, the interviewees were not involved in any evaluation of automated GUI-based system or acceptance testing tools.

High costs for the introduction, execution, and maintenance of tests was stated as a problem by interviewees at all teams but TE1. The interviewee for TA1 and TB1 stated that there is always the possibility to do better testing. However, the costs of doing so are the problem. Additionally, he stated that both TA1 and TB1 do very good testing compared with the state of practice. In Company C, all interviewees stated that there was too little time for testing. This was attributed both to the high costs of software testing and to the attitude towards testing within the company. It was brought up during the interviews that Company C had moved from a plan-driven process to Scrum. However, many developers at Company C still saw testing as an optional practice in case there was time left after development.

With regard to GUI-based acceptance testing, customer involvement was mentioned to be problematic by teams TC1, TC2, TD1, TE1, and TF1. While TE1 already involves customers in their acceptance testing process, they would like to increase this involvement. The interviewees stated that it was usually not easy to convince customers to get involved. Additionally, several interviewees mentioned that customers typically want a qualitative product, but are not interested in the quality assurance process. Team TC1 at Company C has mostly users within the same company. The interviewees in team TC1 mentioned that it is difficult to involve these internal users as they are usually busy with their own projects. Both Company D and F are market-driven companies. Therefore, they face the problem of choosing representative customers. The interviewees in TF1 mentioned that, in order to get involved with the company's testing, customers would often expect their own feature requests to be prioritised in return. One interviewee in TF1 stated that there should be a bigger focus on the problem of customer selection in research. That is, which customers or customer representatives you select for testing purposes in market-driven companies.

Both teams TC2 and TE1 stated that test data for automated testing is difficult to handle. In the case of TC2, the difficulty was in setting up the test system with the right test data. On the other hand, the interviewees in TE1 stated the problem with regard to the whole test system. They stated that it is difficult to set up a test system with the "right test data and a proper back-end". Interestingly, the problem was never mentioned with respect to manual testing.

Finally, the issue of testing legacy software was brought up in TB1 and TF1. In TB1, the problem was related to the test scenario documentation. Their system has existed for over twenty years. In this period, test scenarios have gradually been added. As a result, many test scenarios can no longer be traced to requirements. Therefore, it is difficult to decide if these scenarios are obsolete or not. The interviewee in TB1 stated traceability of requirements to test scenarios as one of the main challenges in the company's testing practices. In TF1, the issue is related to introducing new tests for a legacy system. Due to time constraints, the company focussed on developing new functionality instead of testing in the last years. This practice resulted in a system that is generally hard to test. Now, the team has more time for quality assurance but at the same time has difficulties introducing testing.

C. Discussion

In the studied cases, there is more testing on system-level than expected prior to the study. The testing is mostly conducted in the form of manual system testing through the

GUI. TA1, TB1, and TC2 have extensive manual system testing. Team TD1 performs manual system testing in waterfall projects. TF1 does manual exploratory system testing. However, TF1 does not have any documented test cases. Customer interaction exists only rarely for testing purposes. In the studied cases, customer interaction in testing was mostly non-formalised, improvised testing with no defined or documented test cases. In teams TC1, TC2, TD1, TE1, and TF1, it was stated as one of the main problems to involve customers for testing purposes. In contrast to common research in the area of acceptance testing, none of the teams used a tool which facilitates acceptance test scenario creation, such as FIT.

In most studied cases, test scenarios are elicited by developers or testers from their domain knowledge. Often, vague scenarios like user stories are extended to full test scenarios by the developers or testers. None of the studied teams used models to generate or create scenarios. This is in contrast to the amount of research in Model-Based Testing.

Automation of GUI-based system and acceptance tests exists, but various problems remain. Mostly, these problems are attributed to missing time, problematic handling of test data, or missing tool support. Maintenance of manual test suites is working well within the studied companies. Interviewees in two of the studied cases stated that maintenance of automated test suites will get problematic once many test cases exist. However, this was only their opinion without any experience from previous projects. Actually, maintenance of existing automated test suites within the studied teams seems to be efficient. This can be attributed to a well-structured test suite as in TB1, or to test suites with few test cases as in TE1. In the studied cases, GUI-based system and acceptance test suites usually consist of linear test cases. Logical connections between test cases are used regularly. However, it was stated that these connections 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 fails.

Hence, there is a large variety between the studied cases with regard to how formalised the testing is performed and which techniques and tools are used. This complements the findings in Itkonen et al. [15], that “testers, in practice, apply numerous techniques and strategies during test execution and do not mechanically rely on test documentation.”. As most teams use manual testing, the need for more research on manual GUI-based system and acceptance testing can be seen. This complements the findings in Itkonen et al. [15] and Berner et al. [6]. Interestingly, only in TA1 and TB1, manual and automatic GUI-based system and acceptance test suites existed at the same time. Additionally, teams who wanted to introduce automated testing usually only saw the possibility to replace the manual test suites entirely with automated test suites. The option of *balancing* automated and manual testing, as described in Ramler and Wolfmaier [24], was only considered once problems with automated testing tools arose. The fact that existing automated testing tools still show deficits and that most of the studied companies do manual testing complements the statement in Berner et al. [6] that manual testing can not be fully replaced by automated tests. Additionally, the study shows that this also applies to GUI-based system and acceptance testing in particular. However, some of the problems brought up by interviewees during the study also show that the expectations towards test automation are too high. This result supports the findings in Berner et al. [6].

The recently introduced approach at TB1 for automated GUI-based system and acceptance tests looks promising. The team has extensive GUI-based system and acceptance testing with interconnected test cases. An overall graphical structure is used to depict the connection between test cases. On the other side, especially TE1 manages to maintain a high quality even with few test cases and a lightweight GUI-based system and acceptance test suite. Here, only the most important scenarios are implemented as automated GUI-based system test cases. Acceptance testing is done manually by the customer in retrospective meetings. For bug detection, a broad code coverage is reached at unit testing level instead of using system-level tests. This approach could be a good alternative to extensive and costly GUI-based system and acceptance testing for many software projects. The approach can be related to the aspect of ”Just-enough acceptance testing“ described by Stolberg [27].

V. Conclusions

This paper presents the results of a multiple-case study performed with different teams in six software developing companies. The aim of the study was to investigate the current state-of-practice of GUI-based system and acceptance testing in industry. In order to acquire a broader view, software development companies with different characteristics were chosen. The data collection was performed through twelve semi-structured interviews, answering the study's two research questions. Research question RQ1 was *How is GUI-based system and acceptance testing performed in industry?*, and was answered by the following findings:

1) Manual GUI-based system testing is widespread.
2) Formalised GUI-based acceptance testing is not common.
3) Automated GUI-based system and acceptance testing exists mostly on a small scale.

The level of formalism of the test processes differed substantially between the different cases. With formalised testing, the authors refer to the existence of documented test cases and an established test process. Furthermore, only in two cases manual and automated GUI-based system and acceptance test suites could be found at the same time. In addition, academic trends such as Model-based Testing or easy-to-use tools for acceptance testing could not be identified in any of the studied cases.

The second research question RQ2, *Which are the typical problems related to GUI-based system and acceptance testing in industrial practice?*, can be summarised as follows. Observed problems with respect to GUI-based system and acceptance testing at the studied cases are:

1) Limitations of automated testing tools.
2) High costs of testing in general.
3) Introduction of customer involvement.
4) Handling of test data.
5) Testability of legacy software.

These findings also reflect problems addressed in other manual and automated software testing research [6]. The study therefore shows that many problems still exist in industry and in particular in the area of GUI acceptance and system testing. A problem of specific interest is customer involvement, which was pointed out as difficult by almost all interviewees. A second problem was test data generation for automated tests, but not for manual tests. This is an interesting finding that warrants further research.

The presented study shows current approaches and problems in the area of GUI-based system and acceptance testing. It shows that there is still a lot of room for improvement in the practices used in industry. Additionally, the results show that some areas of GUI-based system and acceptance testing, such as manual testing, are currently under-represented in academic research and therefore warrant more attention.

Two different, promising testing approaches were found. In one team, extensive GUI-based system and acceptance tests with interconnected test cases was used, while another team used only a few system test cases and a lightweight GUI-based system and acceptance test suite. These two approaches could perceivably complement each other and help to improve the testing practices in software developing companies, e.g. during different phases of development when the developed product is more or less mature. For instance, the extensive testing approach could be implemented in large projects in later phases, whereas the lightweight approach is suitable for smaller agile projects and in the initial phases of large projects. However, this study does not answer which approach is better in which context. Thus presenting an area of manual and automated system and acceptance testing that warrants more research.

ACKNOWLEDGEMENT

We would like to express our gratitude to all participating individuals and their employing companies for contributing to the data collection and therefore making this study possible.

REFERENCES

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

[2] "Fitnesse," http://fitnesse.org/, Sep. 2012.

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

[4] "Jellytools," http://wiki.netbeans.org/JellyTools, Jan. 2013.

[5] E. Alégroth, R. Feldt, and H. H. Olsson, "Transitioning manual system test suites to automated testing: An industrial case study," in *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, ser. ICST '13. IEEE Computer Society, 2013.

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

[7] L. Briand and Y. Labiche, "A uml-based approach to system testing," *Software and Systems Modeling*, vol. 1, pp. 10–42, 2002.

[8] E. Börjesson and R. Feldt, "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*, ser. ICST '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 350–359.

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

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

[11] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *Software Engineering, IEEE Transactions on*, vol. 28, no. 2, feb 2002.

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

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

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

[15] J. Itkonen, M. Mantyla, and C. Lassenius, "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.

[16] M. Jovic, A. Adamoli, D. Zaparanuks, and M. Hauswirth, "Automating performance testing of interactive java applications," in *AST '10: Proceedings of the 5th Workshop on Automation of Software Test*. New York, NY, USA: ACM, 2010, pp. 8–15.

[17] D. Kundu, M. Sarma, D. Samanta, and R. Mall, "System testing for object-oriented systems with test case prioritization," *Software Testing, Verification and Reliability*, vol. 19, no. 4, pp. 297–333, 2009.

[18] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville, "'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.

[19] G. I. Melnik, "Empirical analysis of executeable acceptance test driven development," Ph.D. dissertation, 2007.

[20] A. M. Memon, "GUI testing: Pitfalls and process," *Computer*, vol. 35, no. 8, pp. 87–88, 2002.

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

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

[23] C. Nebut, F. Fleurey, Y. Le Traon, and J. Jezequel, "Automatic test generation: a use case driven approach," *Software Engineering, IEEE Transactions on*, vol. 32, no. 3, pp. 140 – 155, march 2006.

[24] R. Ramler and K. Wolfmaier, "Economic perspectives in test automation: balancing automated and manual testing with opportunity cost," in *Proceedings of the 2006 international workshop on Automation of software test*. New York, NY, USA: ACM, 2006, pp. 85–91.

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

[26] H. Srikanth and S. Banerjee, "Improving test efficiency through system test prioritization," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1176 – 1187, 2012.

[27] S. Stolberg, "Enabling agile testing through continuous integration," in *Agile Conference, 2009. AGILE '09.*, aug. 2009, pp. 369 –374.

[28] A. Tinkham and C. Kaner, "Exploring Exploratory Testing," in *STAR East conference*, 2003.

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