



CHALMERS
UNIVERSITY OF TECHNOLOGY

Visual GUI testing in Continuous Integration

How beneficial is it and what are the drawbacks?

Master's thesis in Software Engineering

Arvid Karlsson, Alexander Radway

Visual GUI testing in Continuous Integration
How beneficial is it and what are the drawbacks?
ARVID KARLSSON, ALEXANDER RADWAY

© ARVID KARLSSON, ALEXANDER RADWAY, 2016.

Supervisor: Emil Alégroth, Department of Software Engineering
Examiner: Eric Knauss, Department of Software Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in L^AT_EX
Gothenburg, Sweden 2016

Visual GUI Testing in Continuous Integration
How beneficial is it and what are the drawbacks?

ARVID KARLSSON

ALEXANDER RADWAY

Department of Software Engineering

Chalmers University of Technology and University of Gothenburg

Abstract

Continuous Integration (CI) is a well-established process within software development, used to prevent problems with integration and to improve software quality. One part of the process is to test the code-base with automated tests and usually low-level tests are used for this purpose. Visual GUI Testing (VGT) was presented to complement the process of regression testing and has previously been shown to be a robust and flexible enough technique to be used as automated testing in industrial practice. This study shows that high level VGT tests works in a CI process in a small web-development environment with no prior automated testing. It also presents the advantages and disadvantages when introducing VGT as the only automated test technique. Furthermore, it presents that VGT tests can improve the business value of the application and code base, in this context.

Keywords: Visual GUI Testing, Continuous Integration, Automated Testing, High Level Test.

Acknowledgements

First of all we would like to thank our supervisor at Chalmers, Emil Alégroth. He has helped us a lot during this study, especially when we struggled, but also keeping us on track at all times. Furthermore, we would like to thank Magnus who were our supervisor at the case company. He provided us with the all the information and material needed during this study. Lastly, we would like to thank the rest of the employees at the case company for being so friendly and interested in our study.

Alexander Radway & Arvid Karlsson, Gothenburg, August 2016

Contents

List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 Thesis structure and contributions	2
2 Background	3
2.1 Continuous Integration	3
2.2 Testing	4
2.2.1 Black box testing	4
2.2.2 Low-level test	5
2.2.3 High level test	5
2.2.4 Visual GUI Testing	7
2.2.5 Testing in CI	7
2.2.6 JAutomate	8
2.3 Domain Background	8
2.4 Research Methodology	11
2.4.1 Design research	11
2.4.2 Interviews	13
2.4.3 Observations	13
2.4.4 Watercooler discussions	13
2.4.5 Document analysis	13
2.5 Evaluation Framework	14
2.6 Stakeholders	15
2.7 Related Work	15
3 Method	17
3.1 Context	17
3.2 Research Questions	17
3.3 Design research	20
3.4 Interviews	23
3.4.1 Analyzing the interviews	24
3.5 Observations	24
3.5.1 Case 1 observations	25
3.5.2 Case 2 observations	25
3.6 Water cooler discussion	25

3.7	Document analysis	25
3.8	Verifying the data	25
3.9	Applying the data	26
3.10	Implementation	27
3.10.1	Continuous Integration	27
	Continuous Integration with VGT	27
3.10.2	Visual GUI Testing	27
	Disabled features in the System under Test	28
3.10.3	Engineering solutions	28
	Architectural solution	28
	TightVNC	29
	Engineering the test suite	29
	Integration tool	30
4	Results	32
4.1	Qualitative data	32
4.2	Quantitative data	33
	Prioritization	37
4.3	Evaluation Framework	39
5	Discussion	41
5.1	Implication of result	41
5.1.1	CI with VGT	41
	Advantages	42
	Disadvantages	43
	Limitations	43
5.1.2	Impact on the industry and the academia	46
5.1.3	Answers to research questions	46
	RQ1	46
	RQ2	46
5.2	Limitations	47
5.3	Threats to validity	47
5.3.1	Construct validity	47
5.3.2	Internal validity	48
5.3.3	External validity	48
5.4	Reliability	48
5.5	Future work	49
6	Conclusion	50
	Bibliography	51
A	Appendix 1	I

List of Figures

2.1	A flowchart diagram of how an interaction with a GUI component effect the system, the red flow shows an example of an interaction . . .	6
2.2	An overview of the development architecture previous to this study . . .	10
2.3	A flowchart diagram displaying a generic model of the design research method	12
3.1	How the research questions and sub-research questions ties together. . .	19
3.2	A flowchart diagram displaying how the design research was used in this study.	22
3.3	An overview of the content analysis, showing how the data from the transcribed interviews have been narrowed down to conclusions. . . .	24
3.4	A general overview of the triangulation method, used to validate the data	26
3.5	What methods that have been used to triangulate different areas during each cycle	27
3.6	A flowchart diagram of how TightVNC was used	29
4.1	Outcome from the executed test suite.	34
4.2	Statistical data collected when CI was introduced at the organization.	35
4.3	Actual and approximate descriptive values collected during the study. The y-axis shows the time spent in minutes and the x-axis shows the current week.	36
4.4	Mapping of the results to the chosen evaluation framework.	40
5.1	How the company have and possibly will change over time across multiple dimensions.	45

List of Tables

2.1	Characteristics of the system under test	9
3.1	A summary over the conducted interviews.	23
4.1	The result from the prioritization formula, the higher number (1-4)/the more red a feature is, means that the feature needs to be tested more often.	39
5.1	Found advantages, disadvantages and limitations during this study . .	42
A.1	Valuation of the test cases in the four categories from section 4.2 . . .	II

1

Introduction

Software developing processes are constantly changing. The latest trend is an agile method to engineer new software, where an agile workflow emphasizes on making working software, rather than implementing all features in it [1, 2]. A new innovative process that has been developed to follow this agile workflow is Continuous Integration (CI) [3]. CI as a process is not well defined by strict rules as different contexts requires different parts of CI [3], but the common idea behind CI is to prevent problems with integration and to improve software quality by constantly integrating new code into a shared repository [4]. The guidelines in CI states that an automated test suite should be executed on the system under development. This is to ensure the quality of the newly committed code and to ensure that the integrated code has not modified other features in the software i.e. regression testing [5].

An automated test suite in CI is commonly written as unit tests [6]. With unit tests, the code itself is tested, but not the actual behavior of the software since unit tests operate on a low level of the system under test (SUT). Therefore, they do not capture runtime failures [7].

A common persuader to implement CI is to ensure the quality of the code and to set policies that require automated regression testing on new code. However, only low level unit tests will not be able to test the functionality of the SUT fully; this implies that high level tests would be useful as the automated tests, from a test coverage perspective [7]. Previous attempts have included automated unit tests that acts as high-level tests, but they have been too complex to maintain[8]. Other alternatives include Record and Replay (R&R) techniques, [9, 10] but they suffer from the limitations that either the tester needs access to the code base of the SUT or the tools are sensitive to GUI changes. This means that changes to the SUT require the tests to be modified, and as a consequence it leads to a unfeasible maintenance cost to be profitable [11, 12].

To also test higher levels of the SUT, i.e. input through the Graphical User Interface (GUI) Alégroth et. al introduced a new method of automated testing called Visual GUI testing (VGT) [13]. VGT relies on image recognition and emulates user input to the GUI [8, 13, 14]. Early tools engineered for VGT mitigated the problems with previous techniques for high level tests through the GUI. The image recognition made VGT robust enough to changes in the GUI and also removed the need for the tester to know the code base [13]. However, an early limitation with VGT compared to previous techniques is that the engineering time to implement a test suite in VGT was longer than in R&R [13]. This is because all the test cases needed to be written manually which increase the test development costs and they are more error prone compared to tests written with R&R tools [8]. Alégroth et. al have shown that

a newer VGT tool called JAutomate with R&R properties is ready to be used in industry and will mitigate these limitations [14]. But to the authors best knowledge it has not been investigated in a CI environment where tests need to be written and updated continuously.

In this study the goal is therefore to investigate if VGT can be used at all in CI and if this type of automated higher abstraction level tests is profitable in a CI environment. This is achieved by introducing CI with VGT as the only automated test technique in an industrial environment. The effects of introducing VGT will be compared against the total cost, which includes the cost to implement and maintain the tests as well as the execution time of the tests. In addition, the study will analyze how the introduction of VGT and CI will affect different values in the company, e.g. change of processes, development time, errors in the system, etc. Together, the goal is to answer the research questions which can be seen in section 3.2, they will in turn help to answer if it is beneficial with VGT in CI or if the drawbacks are too many.

1.1 Thesis structure and contributions

Chapter 2 describes the theory behind CI and why it is used by developers. It will also present VGT as a concept, the theory behind the technique as well as known benefits and limitations. Chapter 3 describes the context in which the thesis was conducted, what research questions that were answered, the scientific method used and processes introduced to answer these questions. There will also be an explanation over how the study was conducted and major decisions made by the authors. Chapter 4 describes the different kinds of data and result obtained in the study. In chapter 5 a discussion of the result, threats to validity and the areas where future work is needed will be presented. Finally chapter 6 summarizes the study and presents the authors conclusion.

2

Background

This section covers the technologies and processes together with related work that has been used throughout this thesis work. Section 2.1 describes CI process. Section 2.2 describes software testing in general while section 2.2.4 describes VGT. Section 2.2.6 describe the software used in order to carry out the VGT and section 2.3 describes the domain in which this study was conducted. Section 2.4.1 explains the scientific method that is used in this study, section 2.5 states the model that is used to evaluate how the introduction of Continuous Integration and Visual GUI testing have affected the company where section 2.6 explains the stakeholders that will be the substratum to the results obtained.

2.1 Continuous Integration

Continuous Integration (CI) originates from eXtreme Programming [4, 15] and is a process used to prevent problems with integration and to improve software quality. This is achieved by doing smaller and more frequent commits to the project repository [16]. By doing so, the risk of errors in the code is reduced because it is less likely to have merge conflicts [5] and the goal is that the project compiles and passes its test suite at all times [17]. Furthermore, when using CI all developers should easily be able to see what changes have been made to the shared repository as well as the status of the system [17]. A synergy effect of frequent commits is that it is easier to achieve this, since uncommitted code is invisible to other developers [5]. Another persuader to adapt to CI is that it can help an individual developer as well as the whole team to deliver new features in shorter time and with higher quality, because of its processes and tools [17].

Introducing CI in the development workflow means that several processes and tools may be introduced [16]. As discussed in “Modeling continuous integration practice differences in industry software development”, the CI process differs in the way it is implemented and used in various projects in the industry i.e. through tooling, processes etc. [3]. However, all projects that comply with a CI process have some common denominators. All code shall be kept in a shared repository using a version control system (VCS), e.g. GIT. An automated build is used to verify that the committed code is working when integrated with the already existing code. The build process can have different triggers for when to execute the server’s build and integration tests, e.g. a polling system that will detect when a change has been made in the shared repository or one/several daily and/or nightly builds [5, 18]. A benefit with the later method, in difference to running the tests after each commit,

is that the developers do not have to spend any time waiting for feedback, instead they can continue to work right away [2]. The drawback is that if errors have occurred, the feedback for these errors are not received until the entire test process is completed [2]. It can also be harder to find the error/errors if several developers have committed their changes before the integration build is executed, since there is a change that more than one area of the repository have been affected with faulty code.

If the build fails, or the implemented tests do not pass, the developer who is responsible needs to correct the error/errors before starting the test process again [5, 16]. The integration build then verifies that all tests adhere to the standard specified for the project, i.e. all test pass the minimum requirement set by the stakeholders [5]. If any test fails to meet the minimum requirement, the developer should receive information about this failure promptly, for example in an email. Furthermore, all tests should be able to be reviewed by the development team as this gives the projects current status. This can be achieved by sharing all the test logs in a shared repository with the development team [17].

There are two important factors that organizations that work with CI have to acknowledge. Firstly, the size of the test suite matters, and secondly, the feedback response time must be considered [19]. If the test suite is too small, potential errors might not be found. If the test suite is too big, feedback may not be given fast enough. When more tests are included in the test suite it will of course take longer time to execute it. This needs to be considered, since the essence of using CI is rapid feedback [17]. This is even more critical if the tests are executed after every check in. In these cases, if the feedback response time is too long, the developers could spend time trying to schedule new tasks whilst waiting for an approval of a passed test suite. To schedule a new task could in some cases be difficult and can lead to developers being inefficient [5].

2.2 Testing

There are several levels of software testing and each level requires a different testing method, depending on the different areas being tested. To fulfill this need, many techniques exists.

2.2.1 Black box testing

Black box testing is useful because in general, the tester needs little or no knowledge of the internal code base or programming experience, to conduct these tests [20]. The term originates from testing with a tester who knows the functionality of the software, but has no knowledge of how the code inside the software works. That is, a tester sends input into a "black box", where the inside is unknown and gives an output back [21]. Depending on the input provided to the SUT, the output can be validated as correct or faulty, and this is a good way to determine the behavior of a function or the SUT [21].

2.2.2 Low-level test

To test the code itself, low-level unit tests could be used [6]. These tests are fast to execute, and they will give quick feedback if a code snippet is working as intended [20]. In object oriented programming languages, a unit can be a class, but could also be smaller snippets of code [22]. This is the lowest level of testing and is used to test one unit of code to ensure that it works on its own [22].

After each unit has been tested they will be integrated with other units. To ensure that the integration of multiple units works as expected, a technique called integration tests are used. Integration tests moves up one abstraction level and is used to test requirements on the software like performance, reliability and functionality [23]. This is done by grouping sets of units and perform black box tests on the interfaces from these units [24]. The purpose of these tests is to see how these units' work and interact together to make sure they reach the functional requirements set for the software.

2.2.3 High level test

A procedure called system testing is applied after the lower level tests to test the complete system. During system testing, the tested components are put together, and the whole system is tested together[25]. The system is tested using a black box approach, i.e. in some cases no or little knowledge of how the code work is needed to conduct these tests. However, the tests aim to verify that the requirements are met, so good knowledge of how the system operates is useful[26].

Another high-level test method is regression testing, these tests are used to make sure that previous features of a system is still operating correctly when a new feature is added [24].

Usually, the final test that is performed on a system is acceptance testing. It is the highest level of tests, and is made in collaboration with the stakeholders to assure that the software meets the requirements set by them [7]. A common denominator for these high level tests is that they are usually tested through the GUI. In difference to low level tests, where a specific test is designed to test one unit in the system [6]. A high level test through the GUI will test several units in a system and several system-components [27], e.g. depending on the system state when a button is clicked, this click can generate different actions, for example an error message or load a new page. This will in turn effect different GUI model components in the system, which effect different units in the system, an example of this can be seen in Figure 2.1. These high level tests can be performed as black box tests as well to help inexperienced testers to perform high level tests on a SUT [13].

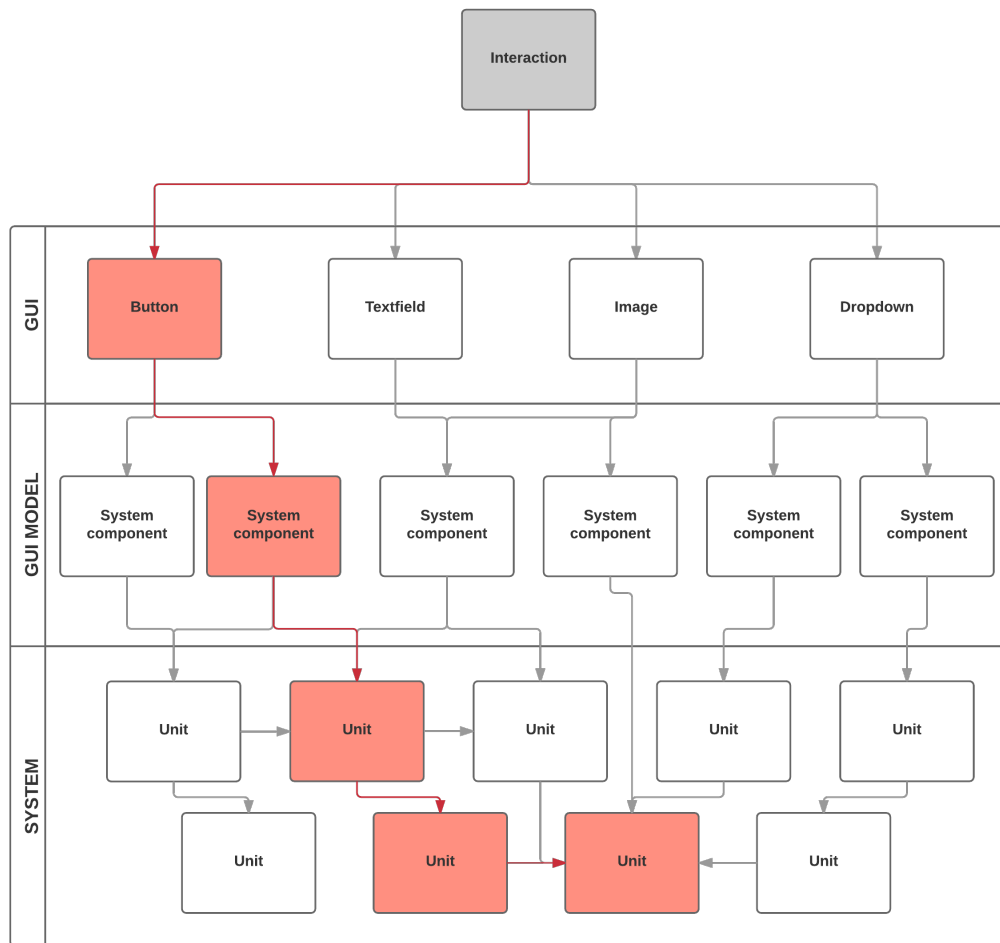


Figure 2.1: A flowchart diagram of how an interaction with a GUI component effect the system, the red flow shows an example of an interaction

High level tests have traditionally been done manually [7], but manual tests are both expensive and error prone [28]. An atomization of regression tests is useful because companies are pushed to deliver software with faster time-to-market and at the same time expected to deliver software with higher quality [28]. To achieve this, companies can remove the error prone and time consuming manual tests and replace them with more agile and substantial automated tests [28]. A technique that can potentially help to reduce both the amount of errors and the cost is record and replay (R&R) [9]. It is a technique where a user interacts with the system under test (SUT) while the interaction is recorded. This recorded script can then be replayed on the SUT automatically to perform regression testing. R&R can be performed in multiple ways. One method is to make references to the SUT's backend, another is to record the exact coordinates and actions made on the SUT's GUI. By recording the actions performed by a user, an automated test suite can be engineered quickly compared to writing each test case on its own. However, R&R is not robust enough to be an automated test technique for the GUI [8]. An R&R test suite based on coordinates will not be able to handle if GUI components are moved [8, 14]. However, an earlier technique to test on the GUI itself, but that is more robust compared to R&R and

unit tests used on a high level is called, component based GUI testing [29, 30]. The technique access the code itself to find the components on the screen. This means that the test ignores how it appears on the GUI and instead analyze a code snippet to verify a component [30]. It will mitigate the problem of GUI changes, but changes in the code will still require an update of the test suite. Another problem with this technique is that it is not what is visible on the screen that is tested, but rather what is coded to be on the screen. If a component is placed off the screen, component based GUI testing would find that component. In relation to R&R, component based GUI testing is more robust, but the development time for a test is much longer. To develop a test suite in component based GUI testing, knowledge about coding and the code inside the SUT is required. To circumvent and mitigate the limitations of earlier techniques such as lack of robustness and minimal flexibility, a technique called Visual GUI Testing has been presented by Alégroth et. al [13].

2.2.4 Visual GUI Testing

The concept Visual GUI Testing (VGT) was presented by Alégroth et. al to complement the process of regression testing i.e. complementing the manual regression tests through the graphical user interface (GUI) with automated tests [13]. Tools developed for VGT uses an image recognition technique to test what is actually displayed on the screen, rather than pre-recorded instructions linked to coordinates or references to the back end [8, 13, 14]. These actions can vary, either they are direct references on the GUI or they are GUI bitmaps that have been identified by the image recognition inside the tool [13]. The output is then recognized with the help of image recognition and measured against one or several expected outputs. A benefit of this is that, VGT does not require the tester to have any knowledge of the code inside the SUT, i.e. a black box test technique can be adapted. A result of the technique is that VGT manages to find a GUI component after it has been moved or if the components color or contrast changes, but not if it has been placed off screen [8]. VGT will also be able to perform successful tests through the GUI even if the back end is changed but the functionality is the same. In comparison to R&R and component based GUI testing, a test suite in VGT will require less refactoring to adapt to changes. These properties could make VGT a more robust and flexible technique to use as automated high level tests, rather than R&R and component based GUI testing as well as a cheaper alternative from a maintenance point of view [13, 14]. A drawback with a test suite engineered in VGT is that it is more expensive to develop than a test suite in R&R [8].

2.2.5 Testing in CI

Test automatization is one of the few mandatory steps in CI [3]. The automated test suite in CI has traditionally consisted of low level unit tests [4, 6]. This implies that even if the tests pass, high level tests like regression tests and acceptance tests have to be done manually to ensure that the software works as intended [7]. To minimize the amount of manual regression tests needed, Alégroth et. al introduced a new technique called VGT [13]. An automated high level test suite would capture more

kinds of errors than unit tests, but most tools are still pretty new on the market and a bit immature, which is one of the limitations with VGT [8]. As a consequence, the initial cost to engineer a test suite in VGT is expensive compared to R&R. A tool that let the user record a test in a similar way as R&R, but still contains the stability of VGT is JAutomate, which is a platform independent commercial tool for VGT. As a consequence, VGT tests written in JAutomate will be cheaper to engineer and maintain [8]. This is a property that is highly desirable when engineering tests for CI; the code is updated daily, and to spend too much time on engineering and maintaining a test suite would minimize the profit gained from CI [17].

2.2.6 JAutomate

In a published paper presented by Alégroth et al. [8] JAutomate was compared against other GUI testing techniques and the result implies that JAutomate, this in combination with a valuation of other VGT tools, made the authors take the decision to use JAutomate as the tool for the automated VGT tests in a CI process. A major advantage is that JAutomate can be used in very different environments and it can be executed on any operating system. This gives the flexibility needed to adapt the tool to the context in need. It also relies on two algorithms for image recognition, where it runs one algorithm for color and one algorithm for contrast [8]. This in turn gives a strong image recognition algorithm that is stable enough to be used as the automated tests in CI to acquire the standard the authors want to achieve. The tool also generates a test report containing data from the tests that was executed. These reports are well structured and make it easier for developers to identify where certain tests have failed and what action that could not be completed. Another positive attribute with JAutomate is that it is also backwards compatible, so there is a little or no risk for a test suite to be outdated if a new version is released. Finally, JAutomate handles the SUT as a black box, so the testing tool do not require any knowledge about it, e.g. its internal architecture, components or the development language used in the SUT [8].

Alégroth et. al found that JAutomate can be operated by novice test users, so there is no need for an experienced programmer to engineer the test suite [8]. This will be useful if a team of testers without a good knowledge of the SUT shall engineer the test suite [21, 31]. The tool also has the possibility to map manual test cases to automated test cases with the relation 1-to-1 [8]. This allows high level regression tests to be executed more frequently and is something that is needed if high level tests shall be used in CI [17].

2.3 Domain Background

This study was done in collaboration with a global distributor of wireless components, e.g. Cellular and Wi-Fi modules, antennas and access point boards. The company is a Swedish, small-sized company with fourteen employees and have a development team of four developers, including the authors of this thesis.

The target of the study was on the company's webpage, which consists of two parts, where one part is the interface towards the customers. This interface is an

e-commerce website where customers can search and order products, find technical documentation and software related to these products, read an FAQ, create tickets for technical support, etc. The other interface is an admin portal, here employees can administer the site and use features for daily operations. A selection of the features includes, updating information about the products, adding new products, create tickets for support cases, creating and sending quotations to a customer, projects can be built and new items can be added to the FAQ.

The logical language used on the website is PHP, the database used is MySQL and to make the communication between the site and the database easier an architectural solution called active record is used. The content management system, or CMS, is built from the ground up and no ready-made CMS application is used. This means that the system is very flexible and a lot of custom made features and functions can and have been made. Furthermore, scripts implemented with Ruby and NodeJS exists to make it easier for developers when compressing code to optimize it for the web.

Language	Lines of code	Controllers	Tables
PHP	321.233	87	-
HTML/JavaScript	308.870	12	-
Ruby/NodeJS	23.576	-	-
MySQL	-	-	82

Table 2.1: Characteristics of the system under test

The developers at the company have a mix of computers running OSX and Linux. To have as similar runtime environment as possible between them and to have consistent behavior of the webpage, each developer has a vagrant box installed [32]. Vagrant is a tool that provisions a thin client of a shared virtual machine [32]. This virtual machine is configured to mimic the environment of the live hosting server, which is an Ubuntu server 14.04. Furthermore, the development team uses a shared GIT repository to share the code and as the version control system (VCS). Finally, to deploy the latest commit to the mainline Fabric is used, an overview of this architecture can be seen in figure 2.2. Fabric is a command-line tool used to streamline and automate SSH deployment between services. Previous to this study the company had no automated test nor was there a CI process, i.e. all previous testing were manual regression and acceptance testing. This means that this study also, in addition to the capabilities of VGT, investigates how the process in software development changes when moving from a development procedure with no CI and no automated tests implemented, to working with a CI process that has an automated test suite in the form of automated VGT tests.

2. Background

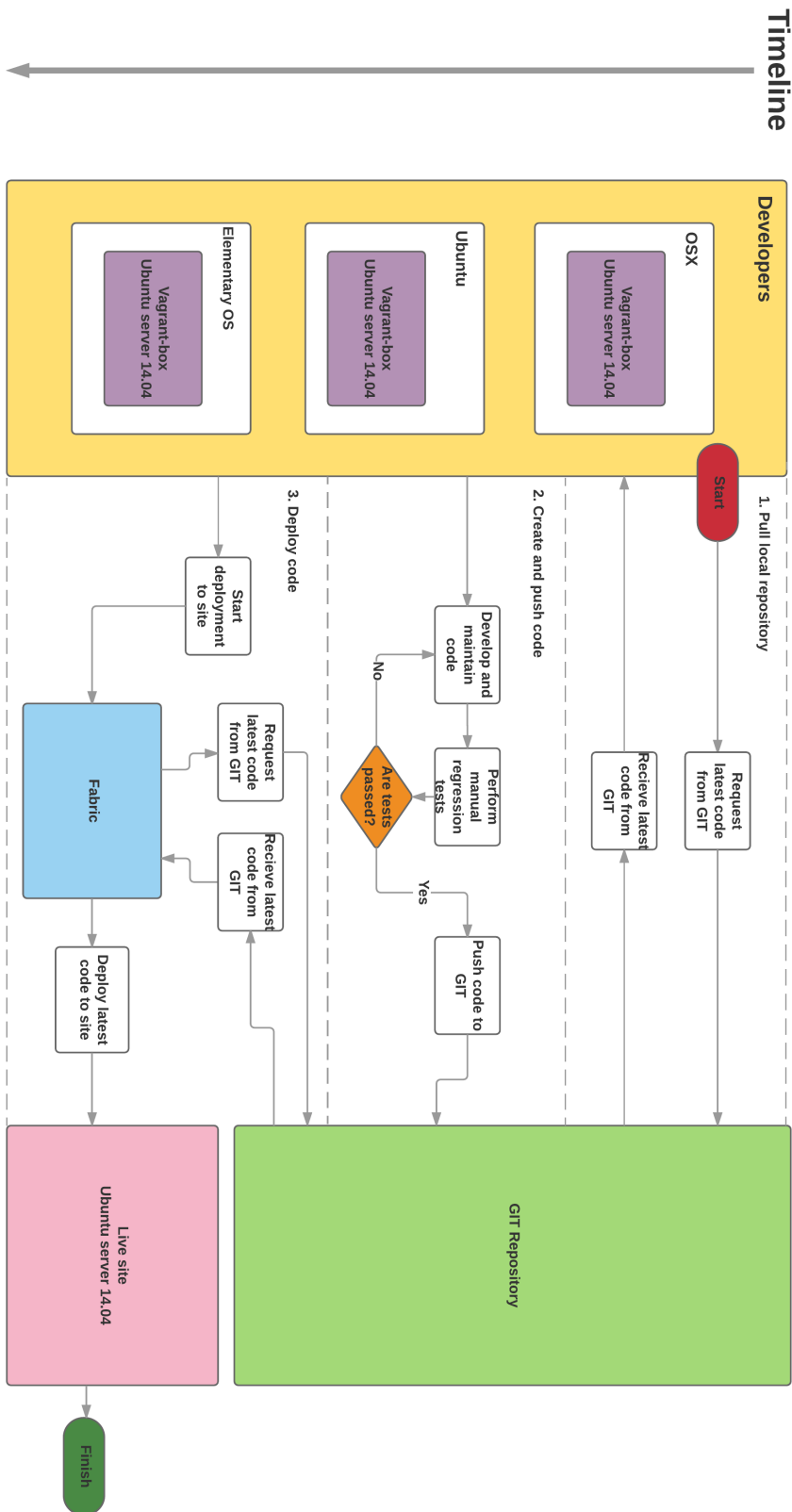


Figure 2.2: An overview of the development architecture previous to this study

2.4 Research Methodology

A study can be performed in various ways. This section will describe the research methodology and techniques that have been used in this thesis.

2.4.1 Design research

The research questions have been answered with the help of a scientific method called design research. Design research is an iterative and cyclical method that focus on the user and lets the researchers take part in the experiment [33, 34]. The initial solutions are based on the observable problems, and the proposed solution is then tested in the current problem space. That is, a problem is defined and a solution is engineered to the defined problem. This process then repeats itself to evaluate and improve the initial solutions [34]. The new solution is then tested and evaluated on the original problem. The solution is then improved on to even better solve the initial problem see figure 2.3.

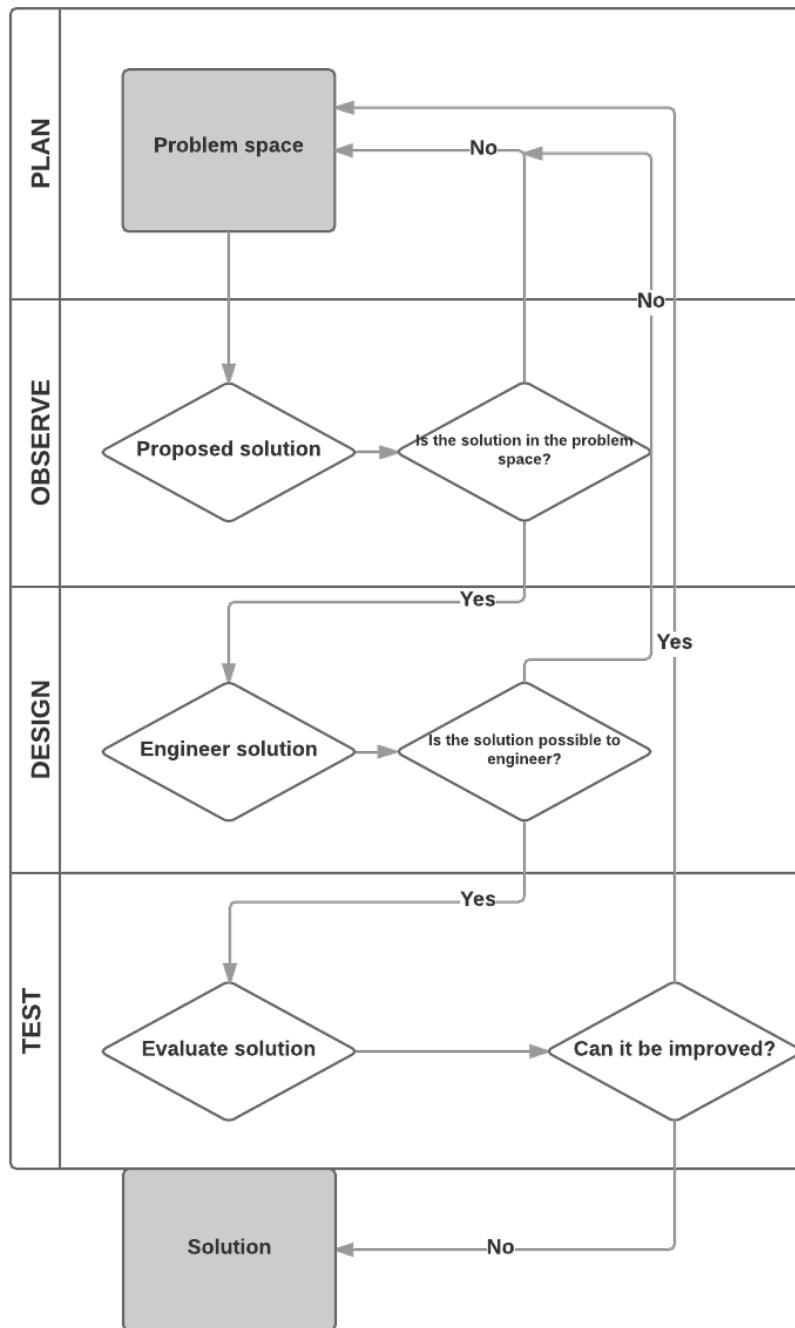


Figure 2.3: A flowchart diagram displaying a generic model of the design research method

2.4.2 Interviews

Interviews are conducted to collect data from interview subjects that is affected by the study [33]. In an interview, the researchers conducting the study shall have questions related to the research questions that they aim to answer, although not the research questions themselves. An interview can be made in various ways and with two different types of questions, open ended questions and closed questions. Open-ended allows the person being interviewed to elaborate and discuss the question while a closed question limits the possible answers [33]. Runeson and Höst [33] presents three ways to conduct an interview in software engineering; unstructured, semi-structured and fully structured. An unstructured interview has more general questions or questions formulated to address concerns rather than being more specific. This makes it possible for the interview to go in the direction that the person being interviewed wants. In a fully structured interview, each question is pre-defined and are always asked in the same order. Semi-structured interviews have the same level of predefined questions as a fully structured interview, but the interviewees may choose to not follow the order of the questionnaire if it is not natural to the conversation.

2.4.3 Observations

Runeson and Höst [33] states that observations shall be conducted if one wants to find out how a certain task is made by people working with software engineering. There are different ways to make an observation depending on the research method selected. For design research, the observations classified as case 1 and case 2 are most suited. An observation of case 1 indicates that the researcher has a high interaction in the process and is part of the team and where the participants also have a high awareness of being observed. In case 2, the researchers also have a high interaction, but the participants has a lower awareness of being observed compared to case 1.

2.4.4 Watercooler discussions

In a workplace, there is usually a photocopier or a coffee machine. These objects becomes a place in the office where people can interact with each other in a more informal way. Fayrad and Weeks [35] refers to these informal interactions between workers water cooler discussions. It originates from the time when the water cooler was common in offices as the place where these informal interactions took place. Saker et. al [36] presented that these types of informal discussions increases information sharing significantly and can be used to get more information from participants in the study, but the statements and answers needs to be triangulated since the information may be biased [37].

2.4.5 Document analysis

Document analysis is a cost effective way to gather information about a company [38]. A document analysis is carried out by first skimming the document, then read the relevant parts thorough and finally interpret the relative parts. The

data collected shall be triangulated to avoid biased selection and complimented with other methods to avoid insufficient details which occurs when important information are not included in documents [38].

2.5 Evaluation Framework

To evaluate how the company as a whole will be affected by the introduction of CI with automated VGT tests, the result cannot be evaluated only by the introduction cost. van der Linden et. al [39] have introduced a multi-perspective framework to evaluate how an organization or product is ranked from a software product line perspective with respect to Business, Architecture, Process and Organization, or BAPO. The framework was originally produced to evaluate a company to see how much they have adapted to software product line engineering; that is, how much of a company's software can be reused to make similar software or a new product for a different market with the same base. In this thesis, this framework has been adapted with the same areas, but with a difference on how some of the areas are measured to allow the authors to capture the multi-dimensional aspects that is needed in software engineering [37]. Each category has a set of values that is measurable in a certain area of a company.

Business indicates how the product will generate profit to the company. This is done by measuring the cost generated by software development and compares it against the profit the product is generating. In software product line terms, the business is how much that can be saved by reusing existing software to engineer new products [39]. In this study, cost is dependent on the other areas in BAPO, while profit is customer satisfaction and revenue by the software.

Architecture in software product line engineering describes how much of an architecture that can be reused. If it is just a plugin that is needed to make a software useable for more things, the architecture scores high from a software product line perspective [39]. This thesis defines architecture as the technical aspect of the software development that explains how to engineer and build the software. It explains the properties of the build environment and the tools used to develop the software. The architecture of the VGT scripts is also explained in this category.

Process indicates how the software development is being made, what roles and responsibilities exist when developing the software in software product line engineering [39]. The definition will be the same in this study.

Organisation is closely related to process. It is the mapping of the roles and responsibilities to the organisations or companies actual structure [39]. The definition in this thesis will be the same as the one for software product line engineering.

When studying one aspect of a company, there is a risk of tunnel vision on a single focus or perspective [39]. To avoid this, Börjesson and Feldt [37] introduced a framework called BAPO/PCF that builds upon the family evaluation framework. They identified the need for a multidimensional evaluation framework when studying

software engineering since, compared to computer science, there are several areas in a company that is affected and not only the technical aspect. The multidimensional perspective gives the opportunity to look how different areas are impacted over time, and how they affect each another. Börjesson and Feldt decided to complement the BAPO framework and use past, current and future to represent the time, where past and current makes it realistic to estimate how the values will change in the future with help of historical data. While they stated that past, current and future it worked in their particular case, other dimensions might be better suited for other research methods.

2.6 Stakeholders

In this study, a stakeholder is a person, a company or an organization that has a concern or interest in the SUT. This will be the developers who are engineering the SUT, the testers who are testing the SUT, the admin users who are using it and the product owner who owns it. These stakeholders need to be considered when evaluating the introduction of CI with VGT as the automated test suite; if the product owner is happy due to increased profit, but the developers cannot stand the new procedure is it a good result?

2.7 Related Work

In order to understand the previous challenges and achievements related to VGT and CI, a literature review had to be carried out. It exists lots of research in both VGT and CI individually but to the authors best knowledge nothing on these subjects together.

A paper from 2008 by Cannizzo et al. [2] tested an application that required continuous tests for performance and robustness. They investigated different tools and strategies in order to find the most valuable solution for automated tests in a CI environment. However, the paper does not discuss anything about VGT with CI even though they discuss a lot of different build techniques.

Another paper about CI from 2008 by Miller [5] analyses the cost of implementing and maintaining the CI server. It also analyses the quality of the code base previous to CI and afterwards. The conclusion was that all development teams moving to a CI process can expect a check-in cost of at least 40% less without losing any quality on the code base. Furthermore, it also concludes that since the project which the study was conducted on was relatively small the advantages of CI did not give as much positive effect as it could have had on a larger project.

A study by Deshpande and Riehle from 2008 [40] investigates if the agile CI processes have had an impact on open source project, i.e. they have analyzed 5122 projects and looked at the size of the code contribution over a project's life-span. The findings in the study show that the size of code contribution by developers in open source project has not been significant affected when using continuous integration processes, within the limitations of the study.

In the field of VGT, Alégroth et al. conducted a study in 2013 [14] that addresses the applicability of VGT as a technique in a real-world context, i.e. when it was used on an industrial grade system. The study was performed when transitioning from manual testing to automated system testing using VGT. In difference to this study, the VGT tool used was Sikuli but never the less the result of their study showed that automated VGT tests have 16 times faster execution time than manual testing. Furthermore, it showed that VGT tests found defects in the system that was previously unknown, due to lack of fully testing the system. It also showed that the company would have a positive return on investment and that VGT is a beneficial and feasible technique for industrial system test automation.

One more paper by Alégroth et al. from 2013 [8] which present JAAutomate a tool used for VGT testing, it also the tool that have been used in this study. The paper presents the tool itself and its benefits compared to other techniques and other VGT tools as well as manual testing. The paper concludes that there is a need for a high-level, cost effective flexible and robust tool for system- and acceptance-test automation and that JAAutomate fills this need in the market space. However, this study only investigates how well JAAutomate is suited as the tool for automated test and not in a CI environment.

Alégroth et al. also presented a paper in 2016 [41] where they investigated the return of investment (ROI) with automated VGT tests. The result showed that automated tests in VGT will give a positive ROI, even if the worst case scenario is achieved. Software testing is between 20% to 50% of the total development cost, and even though the VGT scripts had a high maintenance cost, they did still have a positive ROI when measured against the best, worst and actual case in industry.

3

Method

The first section in this chapter describes the context of this study both from an academic and an industrial viewpoint. Section 3.3 presents the methodology used to conduct the research. Section 3.10.1 and 3.10.2 will define the way CI and VGT was used respectively in this study to be able to use CI with VGT as the only kind of automated tests. Section 3.10.3 describes the technical and engineering solutions used to make CI together with VGT possible.

3.1 Context

This study investigates how the process in software development changed when moving from no CI and no automated tests to working with a CI process that has only VGT as automated tests. By investigating this transition, the authors aim is to investigate if it is possible to have VGT as automated tests in a CI environment and what benefits and drawbacks that may occur with the introduction. The authors will also investigate if this introduction is beneficial for the company described in section 2.3 at all, or if it will have other effects on the company where the study was performed. Usually, a document analysis would have been made to gain knowledge about the domain where the study was carried out. However, the authors already had the required domain knowledge before the study was started. Areas where knowledge was required where the software development process and the structure of the company. Knowledge about the software development process will be needed to identify what methods and processes that are already used, so the transition to CI with an automated test suite can be made as supple as possible. There will also be necessary to have a knowledge about the company's structure and resources. Decisions in this thesis where made due to the structure composed at the company and the amount of resources they had available. If this study is going to be replicated, and the researchers has no knowledge of the domain, a document analysis over the mentioned areas would be recommended.

3.2 Research Questions

As stated in chapter 1 this study aims to investigate if CI with automated tests in the form of VGT will work at all. To investigate this and compile a result, the goal was split into the following research questions.

RQ1:

How does Visual GUI Testing support continuous integration in a small web-development environment without prior automated testing?

RQ2:

How does Visual GUI Testing affect the business value in a Continuous Integration environment, in a small web-development project without prior automated testing?

To help answer these questions, the following sub-research questions were stated.

SRQ1.1:

What are the advantages and disadvantages of Visual GUI Testing in a Continuous Integration environment?

SRQ1.2:

What are the limitations when using Visual GUI Testing as automated tests in Continuous Integration?

SRQ2.1:

How should Visual GUI Tests be prioritized to improve the business value under the time constraints in Continuous Integration?

SRQ2.2:

What are the advantages and disadvantages in business value for an organization when introducing Visual GUI Testing as the test technique in the Continuous Integration process?

3. Method

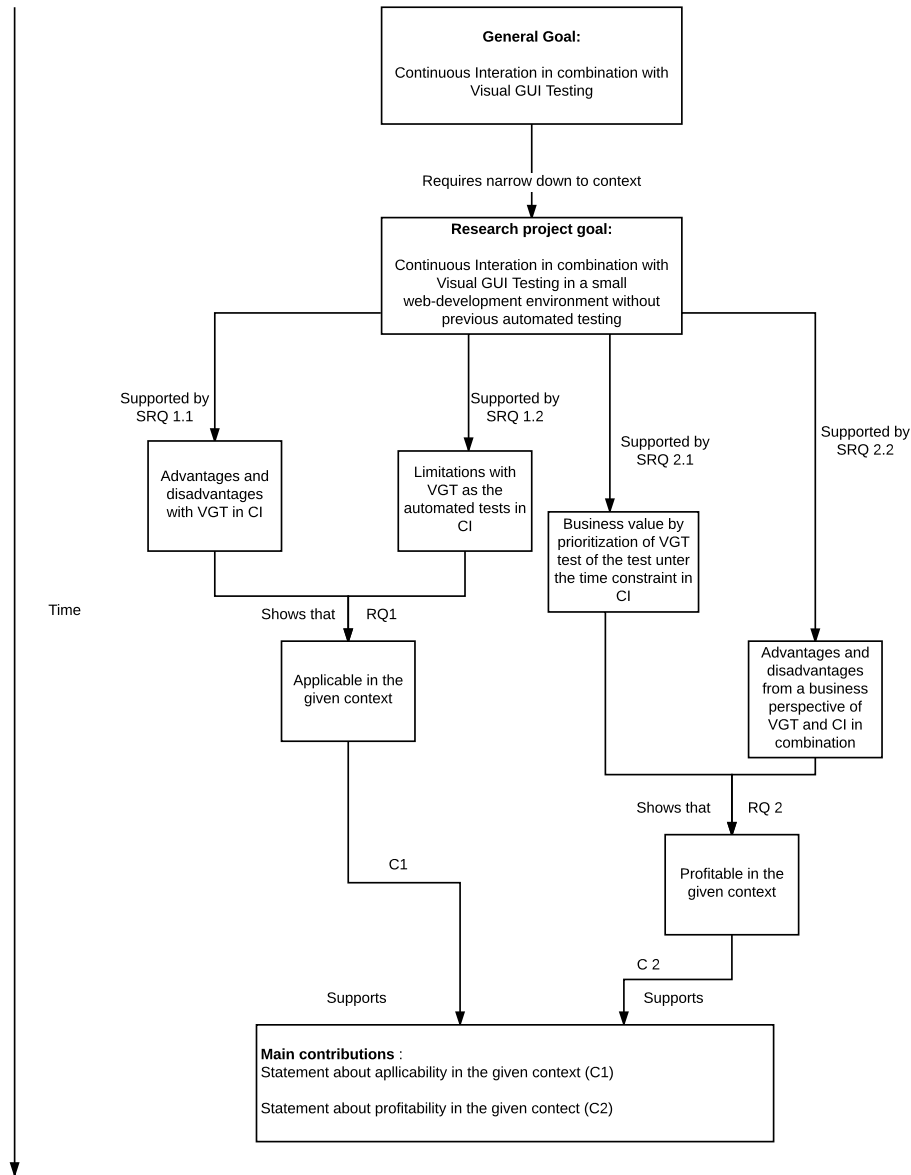


Figure 3.1: How the research questions and sub-research questions ties together.

Figure 3.1 describes how the research questions ties together in the study. Firstly, the sub-research questions are answered. The result from these questions will be the foundation of the main research questions and help to answer these. By answering each sub-research question individually, the goal is to join the result of them to answer the main research questions to conclude if CI with automated VGT tests works in the given context and to see how applicable it is.

3.3 Design research

Design research allowed for the engineering of a test suite and a CI workflow to be integrated at the company. These different cycles allowed several changes and improvements of the developed test suite and workflow over time to investigate what worked well and what did not, since there was a lack of research in this specific area and several open problems had the potential to occur. The iterative process that can be seen in figure 3.2 of design research allows potential problems to be investigated and corrected before any further decisions were made.

One method that is similar to Design Research is Action Research [42, 43]. It is also an iterative model, where a problem is stated, a solution is proposed and evaluated. The main difference is that Design Research is more adaptable to engineering science while Action Research is better for social science. This reflects the areas where the different methods are used. Action Research trends to aim towards a safe solution with more robust and well known technology, while Design Research aims for more cutting edge technology [42]. The partitions also have a slightly different role depending on the methodology [42, 43]. With this in mind, the authors felt that Design Research where a better suite as methodology in this context than an Action Research approach to answer the research questions.

The authors started by identifying the underlying problems that needed to be solved before the research questions could be answered. The main issue was that the development team was out of sync, that is each developer was working more individual without taking into consideration what the others where doing. This led to integration problems and usage of outdated functions in other part of the system. This in combination with no automated tests led to the quality of the code being produced took longer than necessary and lacked the quality possible to achieve.

There where three cycles, the first cycle was dedicated to build up a test suite that had a high enough test coverage and could be executed automatically. First, the test cases needed to be elicited and defined to have a high enough test coverage of the SUT, that is all the features specified by the stakeholders had a test for its function. When the elicitation was done, the engineering of the test suite was started. When the test suite was completed, it was revisited to make sure it covered enough test cases. The revision showed that tests created in the first cycle were in need of refactoring. A reason was that when the initial tests where created the testers experience in VGT where low or insignificant. The testers also lacked knowledge about testing in general. This led to wrongly applied techniques and tests that lacked the required quality to be used in a test suite in CI. In addition to improving the quality of the test suite, we also believe that this practice has helped to mitigate learning bias.

The second cycle introduced CI and the time between the tests and the length of the test were tweaked. The goal of this cycle was to find the time between builds where the ratio considered was the time until feedback was received on the committed code against the time where untested code could be added to the shared repository. It also included to give the development team an understanding over how their development habits needed to change when moving into a CI environment. This included breaking down tasks into smaller pieces and also make sure that the

code committed did pass the automated tests.

The phase where the test suite was refactored was included in the second cycle, because the testers were constantly gaining knowledge regarding both in which sequence tests should be performed and how JAutomate operates. A lot of the initial tests were in need of refactoring to cover more cases and to be more robust. This allows the test suite to grow over time without gaining technical debt or rotten [44].

In phase three, the results of the transition to CI with VGT as the only form of automated tests was gathered. Interviews with developers, users and owners of the SUT were conducted to see how the introduction has affected the different stakeholders of the website.

3. Method

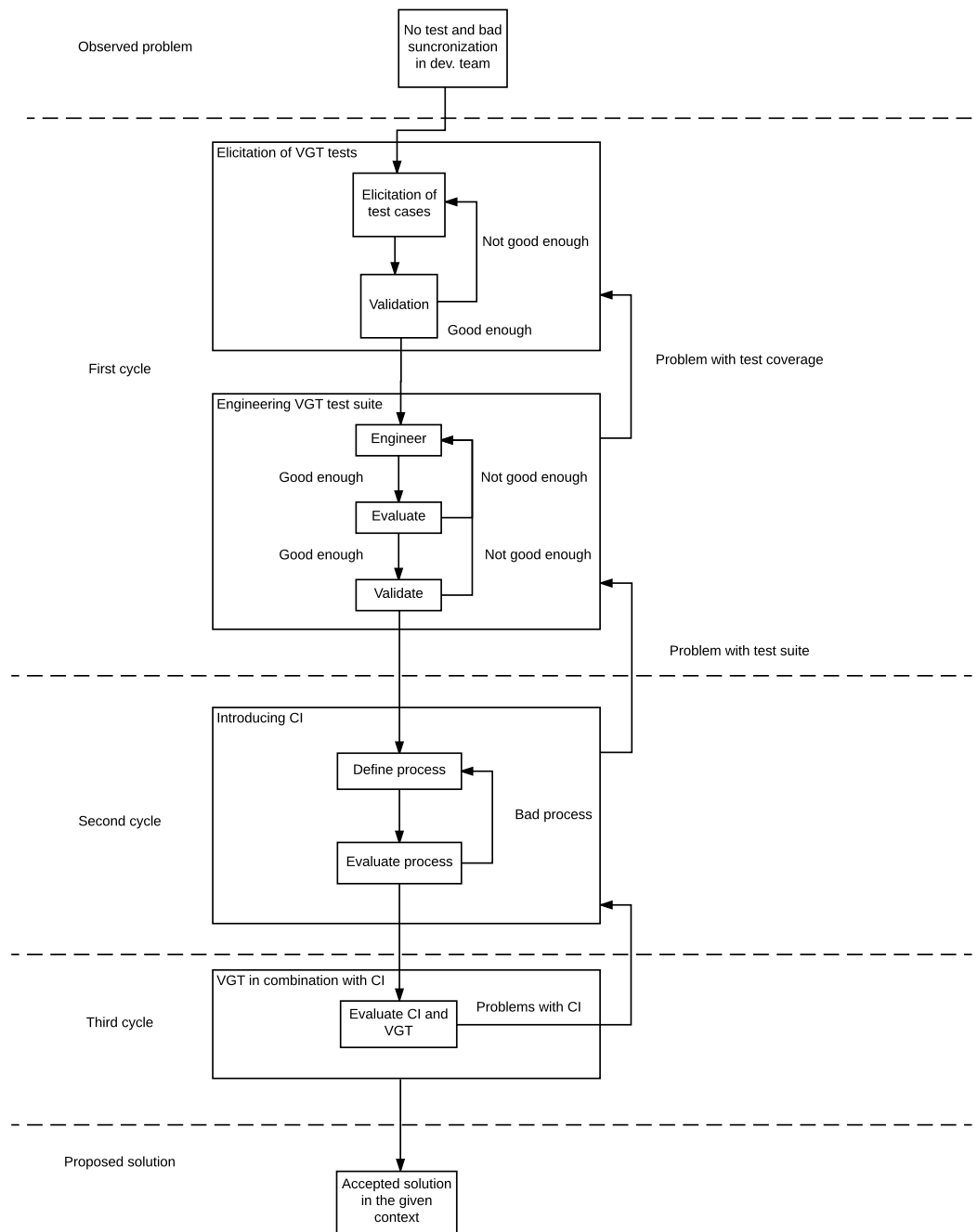


Figure 3.2: A flowchart diagram displaying how the design research was used in this study.

3.4 Interviews

During the course of the study, several semi-structured interviews were held with stakeholders of the SUT. The method of semi-structured interviews is well-established in the software engineering industry [45].

The mean value for the duration of the interviews was approximately one hour, furthermore all of them were held on site at the company of the study and they were recorded. Both the authors conducted all the interviews together, because they believe the benefits presented in the study by Hove and Andy [45] overcome the disadvantages. That is, one led the interview, while the other was more observing and asked additional questions when it was appropriate. After each interview, the recording was transcribed with the intent of making the data easier to analyze and to utilize them into results. The majority of the interviews were held with developers of the SUT, with the purpose to investigate how they perceived the transition to CI with VGT as automated tests, a summary of the interviews can be seen in table 3.1. The authors chose to interview the admin-user and the product owner in order to find out if consequences emerged outside the development team during this study and to get more data on the effect from the introduction of CI with VGT on the daily business operations.

Interviewee / Role	Experience	Average length of interviews	Main reason
DEVELOPER 1	4 years	73 minutes	To get an experienced developers opinions
DEVELOPER 2	8 months	42 minutes	To get an inexperienced developers opinions
ADMIN-USER	1 years	39 minutes	To get a deeper understanding of how an experienced user is affected by the study
PRODUCT OWNER	7 years	64 minutes	To get a deeper understanding of how business values are affected by the study

Table 3.1: A summary over the conducted interviews.

To get a good understanding of how experienced every interviewee was, the first part of the interview focused on their previous experience of using and/or developing the SUT along with their general experience in software engineering. By this, the authors wanted to establish a base of the interviewees knowledge to be able to ratiocinate how much the changes in workflow affected each individual, e.g. if a developer had used CI previously, the adaptation could be easier than one who had not heard of it. In reverse, if a developer had used CI in another way than the process the authors have introduced, the developer may have a harder time adapting. The second part of the interview centered around how the interviewing persons have experienced the transition and introduction of CI together with VGT. Finally, the interviewee was asked to describe if the quality of new or updated features made to the website had been performed, as well as questions related to update frequency.

3.4.1 Analyzing the interviews

As said in section 3.4, the interviews were transcribed using a playback of the recordings. Subsequently, a content analysis was performed to help form a chain of evidence and in the long run to triangulate the content with other results. The procedure used in the content analysis is to map statements from the interview to group them into conclusions. That is, certain words or sentences with similar content were replaced with a code. The coded data were then analyzed to form sub-conclusions. This was then repeated to form even more narrow conclusions and in the end to end up with a few conclusions that have a broad consensus among the interviewees, an overview of this can be seen in figure 3.3.

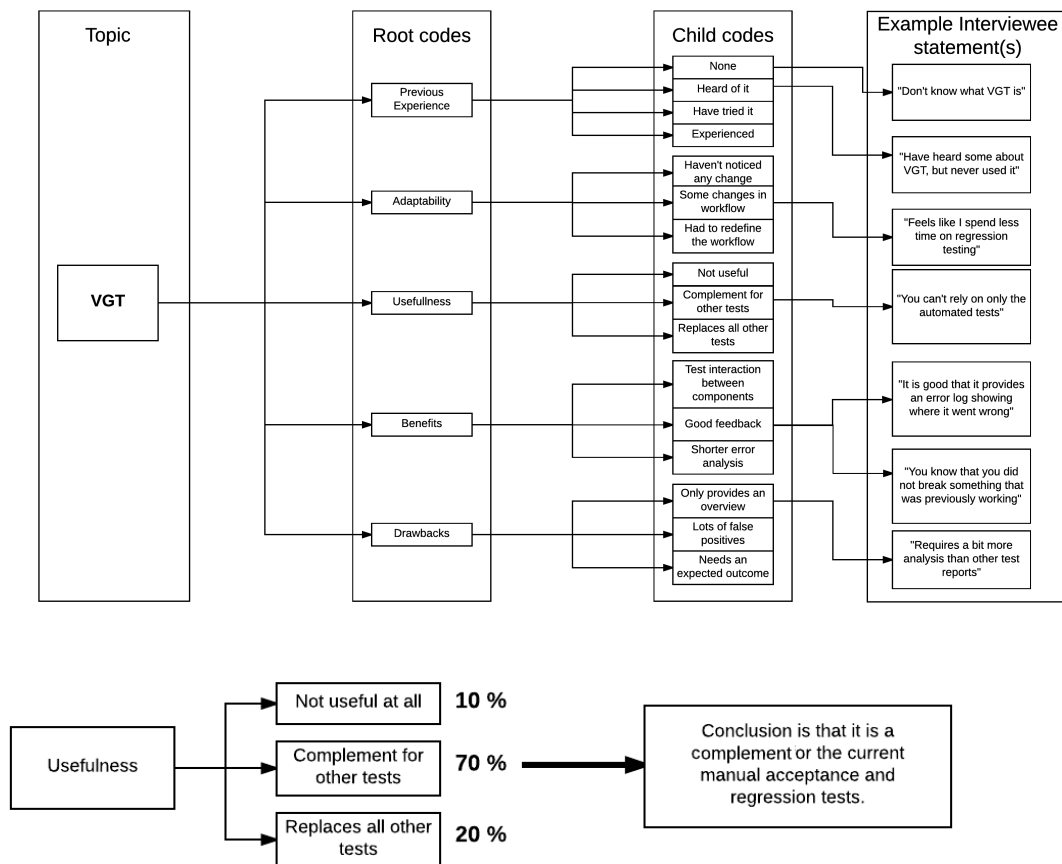


Figure 3.3: An overview of the content analysis, showing how the data from the transcribed interviews have been narrowed down to conclusions.

3.5 Observations

As described in section 3.3 a design research method was used. This gave the authors the possibility to be a part of the study. Because of this, a lot of observations were made. In section 2.4.3 two types of observations was described, case 1 and case 2.

3.5.1 Case 1 observations

To see how well the participants had adopted to the work flow, case 1 observations was conducted. The way the observations were performed was that the developers were observed under one development session of approximately four hours. During the session, the developer being observed performed a think aloud process while the authors observed what the developer did. The main reason for this method was to see that the participants had understood the methods and processes that were implemented, since the participants had to describe what they did and why they did so.

3.5.2 Case 2 observations

The larger amount of observations made was case 2 observations. As described in section 2.4.3 case 2 is when the researchers take part in the study, but the participants do not feel observed. This was done to see how the technique and process worked in industry, and also to gather information of how the developers comprehend working with CI and VGT together.

3.6 Water cooler discussion

In the workplace there were two areas where water cooler discussions arise, the coffee machine and a lounge area with more comfortable furniture. When the personnel were at these places, the authors took the opportunity to ask questions related to CI and VGT from time to time. This was mainly to see how other stakeholders outside the development team were affected by the transition, but also to discuss a bit more informally with the development team.

3.7 Document analysis

To understand the architecture and development process at the company, a document analysis over these areas needs to be made. The analysis was made as described in section 2.4.5. The problem at the case company was that there was insufficient documentation over these areas. The document analysis needed to be complemented with other methods to validate the information obtained.

3.8 Verifying the data

During the course of this study both qualitative and quantitative research methods have been used to get data. In order to verify that the qualitative methods, i.e. interviews, structured observations, various inspections, etc. form a chain of evidence, the authors have used a triangulation method. The reason these methods require triangulation is to remove bias, e.g. in an interview, the interviewee usually tries to be as honest as possible but unwittingly gives their own opinions in the statements [33, 45]. In order to triangulate the data, the information gathered

from the different qualitative methods are grouped, data containing the same acknowledgements form aligned data groups. These data groups are then analyzed to produce unbiased and validated data, see figure 3.4.

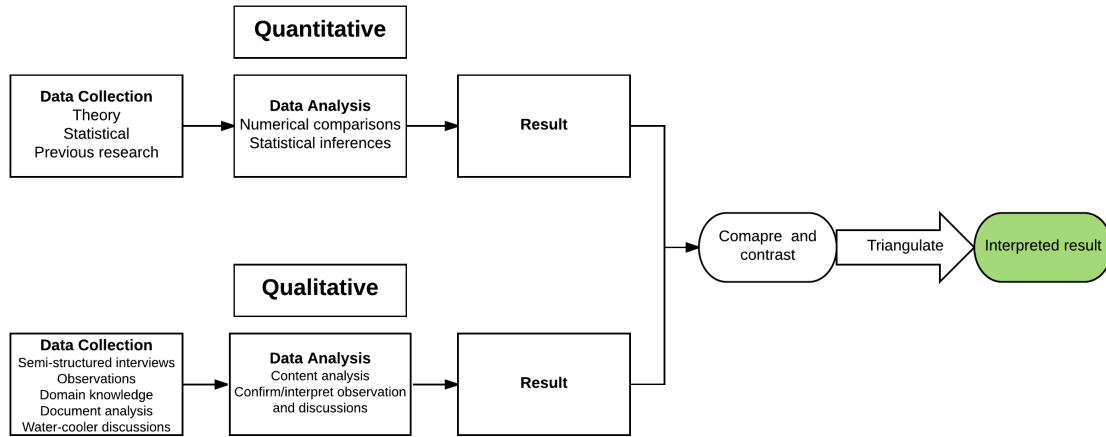


Figure 3.4: A general overview of the triangulation method, used to validate the data

As design research is a cyclical method and new input data can arise at any time during the study, the authors have used the quantitative methods, e.g. data analysis, experiments, etc. throughout the whole study. When analyzing the data obtained in the quantitative methods, 75 percent of the data points have been used to estimate the mathematical models and the remaining 25 percent have been used to verify the models e.g. in figure 4.3.

3.9 Applying the data

By evaluating with a multidimensional framework and investigating how these values change over time during the introduction of CI with automated VGT tests, more potential benefits and drawbacks may be found. The analysed data will be mapped to the evaluation framework described in section 2.5. This have been done by deriving information from the analysed data to investigate how the different areas in the company have been affected by the introduction of CI with VGT as automated tests and to see how beneficial the new process is.

The authors have decided to also use time, but use the different cycles in the design research as the chronological dimension to complement BAPO. This will allow the authors to evaluate the proposed solution from the current cycle in the design research, and also see what effects the selected actions had in this context; it will allow for a creation of a cause effect chain for each individual area between the cycles and also allow for a multidimensional analysis that shows the broaden effect an action have had at the company. The diferent methods that have been used to triangulate each area can be seen in figure 3.5.

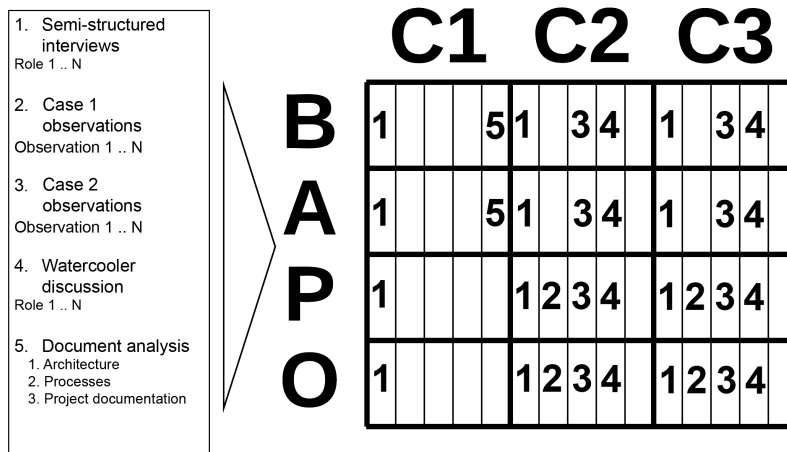


Figure 3.5: What methods that have been used to triangulate different areas during each cycle

3.10 Implementation

This section explains how both CI and automated VGT tests were implemented in the development process. It will also describe changes that needed to be made to get the process and the technique to work at the company.

3.10.1 Continuous Integration

The introduction of CI looks different between different domains[3]. In this section, decisions to make CI and VGT work together as well as changes to the workflow in this context when CI with automated tests in the form of VGT were introduced, will be presented.

Continuous Integration with VGT

The way CI was implemented together with VGT is with daily and nightly builds and tests. It was chosen because the long runtime of the 141 automated tests in the test suite. As a consequence the developers will get feedback of their changes two times per day. The daily and nightly build will allow the developers to get feedback the first thing in the morning and after lunch; they will then have time to investigate potential errors.

3.10.2 Visual GUI Testing

Automated tests written in VGT are different from the low level tests that testers and developers at the company were more familiar with. This section will state some modification and differences that the testers and developers had to adapt to when introducing VGT.

Disabled features in the System under Test

The SUT is sending out emails when different features are used, e.g. when a purchase is being made or a support ticket is updated. In the live system these emails go to the correspondent user or admin, but in the development environment these emails always go to the developers. To avoid all these e-mails to be sent out during testing, the email functionality has been suspended in the development environment. Otherwise mailboxes would be flooded with e-mails and unnecessary traffic is generated.

To allow testing on all features in the SUT, some modifications to certain functions was made. When a new user is registered, there is a verification system implemented called CAPTCHA [46]. This verification system makes it unable for non-human users to create a user. It is done by having an image with letters or numbers showing on the screen. To create a new user, these letters or numbers needs to be entered in a text field before a registration can be completed. Each time the page is reloaded, a new image with letters or numbers is showing. VGT tests cannot handle CAPTCHA due to the image being changed between each execution of the test suite. To circumvent this, the CAPTCHA has been locked to always show the same picture. We perceive that similar context dependent changes are required in any environment. As such, even though these specific changes may not be applicable in another context, they indicate the need for such delimitations in a study of this type.

3.10.3 Engineering solutions

This section presents the technical and engineer based solutions that were used to allow for automated VGT tests in CI.

Architectural solution

VGT needs a reference computer to operate correctly, because difference in resolution between different computers will make the image recognition fail. A reference computer running Ubuntu 14.04 with JAutomate and Git where used to act as the build server. The operative system Ubuntu was chosen because the authors felt more familiar with a Linux environment and also preferred bash as a shell for automated scripting and also because their knowledge of how way files is handled in Linux is greater than in other operative systems. It is also a free operative system, so no upfront investment was needed. As stated in section 2.3 the shared code is hosted on a Git repository hosting server. Therefore, Git is also installed on the reference computer since changes to the web site is pushed and pulled via Git at the company.

The reference computer where given a static IP, in case of reboots, so the testers always knew which IP to establish a connection with. To connect to the reference computer remotely and engineer the test suite a generic version of TightVNC was installed on the reference computer. A VPN connection allows the testers to engineer the tests from outside the network as well, if the testers where not inside the same local network as the reference computer.

To make sure the development environment on the reference computer and the development environment on the developer's local computer are as similar as possible to the server environment hosting the live site, a vagrant box is used as stated in section 2.3. This will allow tests executed in the development environment on the reference computer to have the same behavior as equivalent tests executed on the developers local computers and on the live site.

TightVNC

To allow one computer to connect to another computer remotely, TightVNC was used. This enabled the possibility to use a reference computer; however, the limitation is that only one user at a time can be connected to the reference computer. A flowchart of the setup can be seen in figure 3.6. This software where selected because of two reasons. It had the possibility to be set up as an auto start service, so if the reference computer needed to be rebooted remotely, and the possible to access the reference computer after the reboot. It is also a freeware written in Java, so if necessary it can be used in several environments without any additional cost. Furthermore, the VGT software used in this study, JAutomate have a built in feature similar to TightVNC, but it was not used in this study, due to the authors decision in the given context.

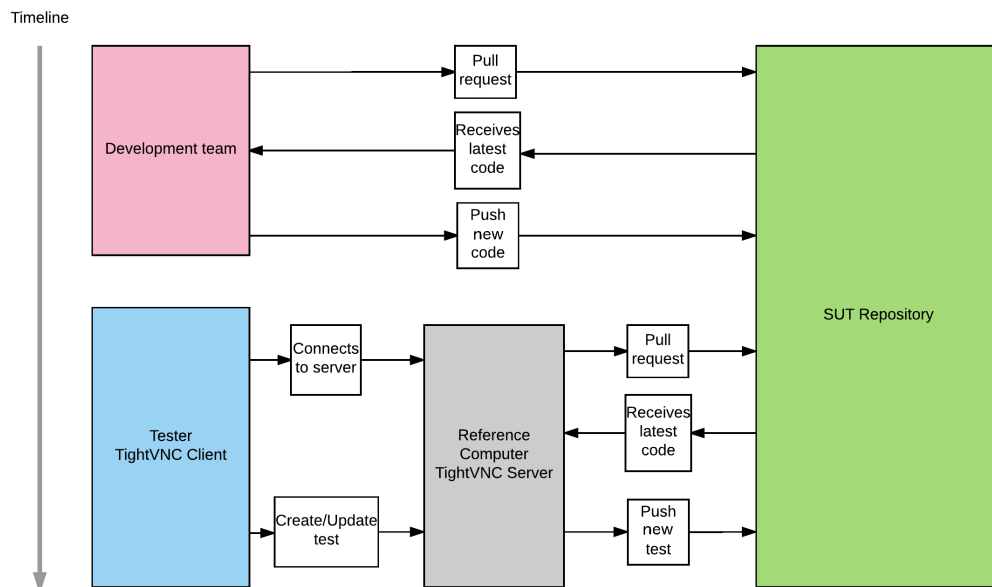


Figure 3.6: A flowchart diagram of how TightVNC was used

Engineering the test suite

The testers who engineered the test suite had no previous experience in writing tests as stated in section 3.3. To gain knowledge of testing the testers did research on testing in general. To transform this knowledge into an automated test suite, testers started engineer tests that were never intended to be a part of the test suite

in JAutomate on their own. This was to force each individual tester to learn the software thorough, and not only know it in theory. It also allowed for beginner mistakes to be addressed early on and not include them in tests intended for the final test suite. After the initial learning stage the testers started development of the automated test suite.

In the first version of the test suite, the tasks where performed in the logical order for which a user would make them, e.g. entering the username before entering the password even if this would mean more cursor movement and scrolling. A refactoring of the tests to optimize movement and input made through the GUI made the run time shorter, but kept the coverage at the same level. Furthermore, the orders of the tests were rearranged to have a more logical order with respect to runtime, i.e. minimize the actions to prepare a test, e.g. minimize log-ins and log-outs. This gave a second version of the test suite that was stable enough to handle changes made to the SUT, covering enough functions and had an acceptable runtime. The refactoring phase also includes commenting the tests to make them understandable to other testers and easier to maintain. It also included a break down of bigger tests to smaller tests with the purpose of making the tests easier to maintain and reuse if necessary.

When the test suite was engineered and executed, the tester realized the SUT behaves as a state machine. The SUT is in a given state under a given time when the test suite is executed. This state machine had to be the base of the logical order of the different tests that is included in the test suite. If a correctly written test is executed on the SUT, but the SUT is in the wrong state, the test will fail regardless. A lot of the refactoring went into placing the individual working tests we wrote in the correct order so they can interact together.

To minimize the run time, testing practices described in [31] where adapted. If a function or feature were tested thorough once in the test suite, it could be reused without need of further testing. An example is login, an action that is executed multiple times. The first test executes multiple cases to ensure that the feature works as intended. If login where needed in a second test, it could be used without a full test.

Integration tool

The daily and nightly builds were executed by running a cronjob which in turn executed a bash script. The bash script pulls the latest version from the Git repository, and executed the automated tests on the SUT. This was done by executing a headless version of JAutomate with the start script as parameter. The reports generated from the tests are then analyzed automatically in the bash script. Each generated report has similar structure, so the bash script takes the content in a generated report and looks for the string stating if the overall test suite passed or failed. Subsequently an email with the result of the executed tests and the reports is sent out to the testers and the developers. Finally, the bash script saves the reports on a shared server so they can be accessed by persons not on the mailing list.

A solution with a bash script was selected over the use of CI software like Jenkins, because the built in functionality of such software would not have increased the benefit in this study, e.g. made it simpler with already build in functions or using

the automated build to compile the code. The authors also had previous experience with bash scripting, so there was no steep learning curve to make the script compatible to the possibility of a potential high-pitched curve to be able to connect Jenkins and JAutomate. There exists documentation on how to launch JAutomate from Jenkins, but the documentation is thin and the authors felt that using bash would mean less time spent setting up an CI environment.

Configuring privileges A problem occurred with the system privileges in Linux during automatic execution of the bash script. This led to several errors related to ownership of created files in Linux and files that were moved with wrong permissions. This problem never occurred during manual execution of the automated tests, so these errors were not easy to investigate and the consequences made the test reports created with permissions the reference computer did not have. This in turn made it impossible for our script to analyze the reports and determine if it was a successful test or a failed one. This was solved by executing the cron job as a normal user, and not a super user which gave access to the result without needing super user privileges.

Another problem was the permissions for a script to access the display, which scripts do not have by default. In this context, scripts were not allowed to open and run GUI components on the display. The script wanted to start a web browser and open the web page to display the SUT, but the script could not proceed with the opening since it did not have access to open GUI components. The StartWeb function in JAutomate would execute, but could not finish the task. This led to the first verification failing, which the authors derived to different reasons than the scripts permission to access the display. After investigations, it was revealed that the permission to the GUI was the root to the error, and with that, the launch of the web-browser by JAutomate was not allowed. To solve this, the following lines were added in the bash script.

```
Export DISPLAY=:0.0
Export XAUTHORITY=/home/USERNAME/.Xauthority
```

This will solve the permission error message that X11 is not allowed to access the display, on a Linux computer. The same issues might not occur when using other operating systems.

4

Results

This chapter will present the result of this study. It contains of two parts, a presentation of the qualitative and the quantitative data gathered during this study.

4.1 Qualitative data

In order to answer the research questions stated in 3.2, data have been collected during the study as described in section 3. With the introduction of CI a lot of previous processes had to be changed for the developers, see section 2.3. All the developers in the team had some previous knowledge of CI before this study, but no one had experience of working with such a process. Two out of four developers had knowledge about component based GUI testing before this study, but no or little knowledge of VGT. The developers had not used software testing at this company before, but one had used ad-hoc acceptance testing previous to this study.

The triangulated data showed that CI with VGT as automated tests do work. The developers are able to continuously integrate code and receive feedback in a time frame that is compatible with the guidelines that exists for CI stated in 2.1. One major advantage stated by all the developers and observed by the authors is, when a continuous testing of the SUT is performed, there is a compose feeling in the development team. Both before and during a deploy to the live site. As one of the developers stated, regarding automated high level tests that is performed continuously, *"...it is useful when you update something, you know that you did not break something that previously working. For example, when updating a library, it is easy to check that it still working..."*. A similar quote was published by Alégroth et. al in "ransitioning Manual System Test Suites to Automated Testing: An Industrial Case Study" *"...It is such a good way to quickly run through and make sure that everything still works..."* [14]. The reason for the contempt feeling in this thesis, is that before this workflow was integrated, the developers had made manual regression and acceptance tests and some test cases were not performed or performed sloppy, either due to time shortage or that conclusions were drawn on what other areas had been affected. The time spent testing was also heavily weighted on new features, and not on older ones to verify previous functionality. There are also statements, indications and observations implicating that fewer bugs were found when new code was integrated, after the transition to CI. One thing to have in mind is that this contempt feeling may be a false sense of security. If the test suite has a bad test coverage, the developers may feel contempt because the system is tested, but in reality the test coverage is to low to actually catch errors that may occur.

The workflow for the development team changed when the introduction of VGT as automated tests in CI was made. The case 1 observations made during ensured that the development team had adapted to the new process and technique. The automated tests could be executed more frequently than the manual regression tests, and this led to less time being spent on trying to find errors in the SUT. Case 2 observations made throughout the study that is backed up by interviews and statements is that less time is spent on manual testing. However, it does still exist to complement the automated tests. The atomization allows for the same regression tests to be executed multiple times a day compared to irregular intervals when the developer had to manually test the SUT.

The developers also had to update the test suite and engineer new throughout the study. Previous studies, described in section 2.2.6, states that JAutomate as a tool that is easy to learn. This result coincides with the authors own experience along with statements from the developers. The possibility that non-technical employees who knows the requirements and test specification can engineer and execute tests was seen as very positive, although no such role existed in this domain.

As stated in section 2.2 a drawback with testing through the GUI is that when a test fails, the root cause to the problem is not shown. This is a property of VGT that was observed since it affects the feedback received from the tests. A statement from a interview is *"It is good that it is possible to see where it goes wrong, but it requires a bit more analysis than other test reports I have seen earlier"*, and this statement harmonizes with observations made by the authors; that it can be hard to recreate the error and a more throughout analysis is required to find the root cause of the error. An observation made by the authors was that the VGT tests have a longer runtime than other test methods used by them before. However, the VGT test still have a shorter runtime than if the same test would be performed manually. An estimation from the authors is that it will take at least twice as long time to go through the whole test suite manually compared to the automated suite, in this context.

Everyday users not included in the development team have stated that they have not noticed the change in any negative way. No new features have been delayed significant from the time frame the developer team have promised, and the consensus among the users is that features developed after the introduction of CI with automated VGT tests have been smoother introduced. There was more complaints towards the development team when features was released previously to the introduction than after, but this may be affected by other factors such as developers gaining skills in engineering new features.

4.2 Quantitative data

One of the aspect that have to be studied in order to answer RQ1 is how reliable VGT is in finding errors, i.e. how many false positives and false negatives exist in the automated suite and how many are actual errors. Figure 4.1, shows the outcome based on 161 runs on the complete test suite which consist of 141 tests. Pass is when no test in the suite have failed or found an error, a false positive is when a test in the suite have failed but no error exists and an actual false is when a test in the

suite have failed, a false negative is when there is a test for a defect function in the SUT, but it does not find the defect. The outcome shows that the statement in section 2.2.4 is correct, i.e. false positive outcome is a regular occurrence in VGT, see figure 4.1.

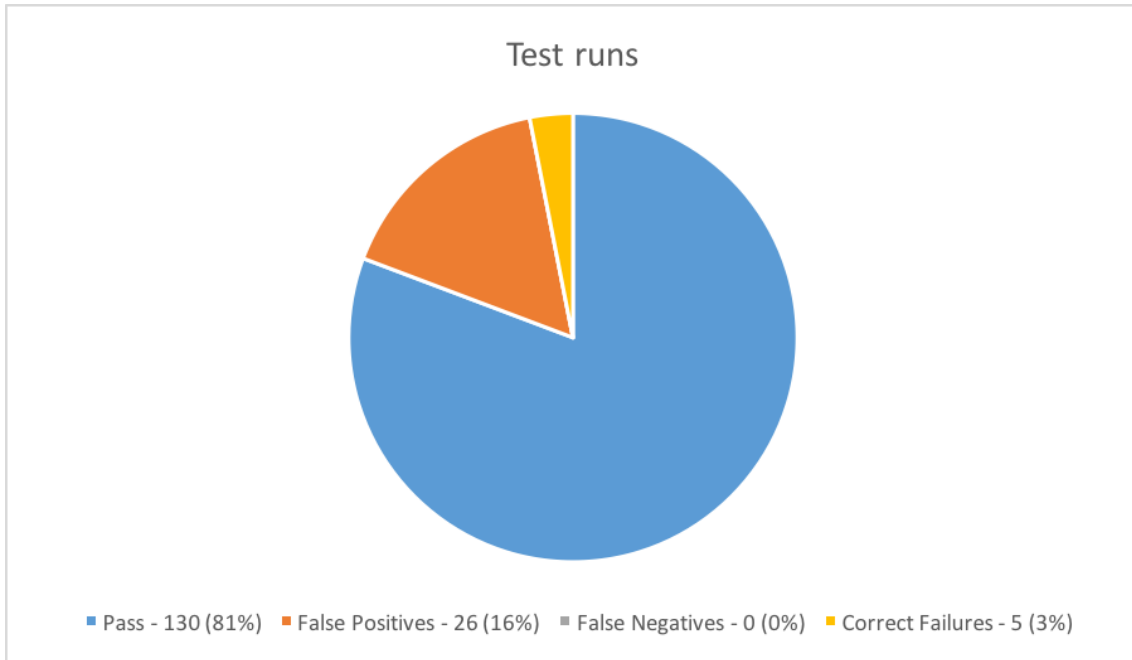


Figure 4.1: Outcome from the executed test suite.

Furthermore, in order to see how the developing processes have been changed statistical data have been collected before and after the introduction of CI. See figure 4.2, the left column shows data prior to the introduction of CI and the right column shows data after the introduction. The data in these tables are concentrated on key aspects of a commit, they have only been collected from the main branch in the shared repository.

4. Results

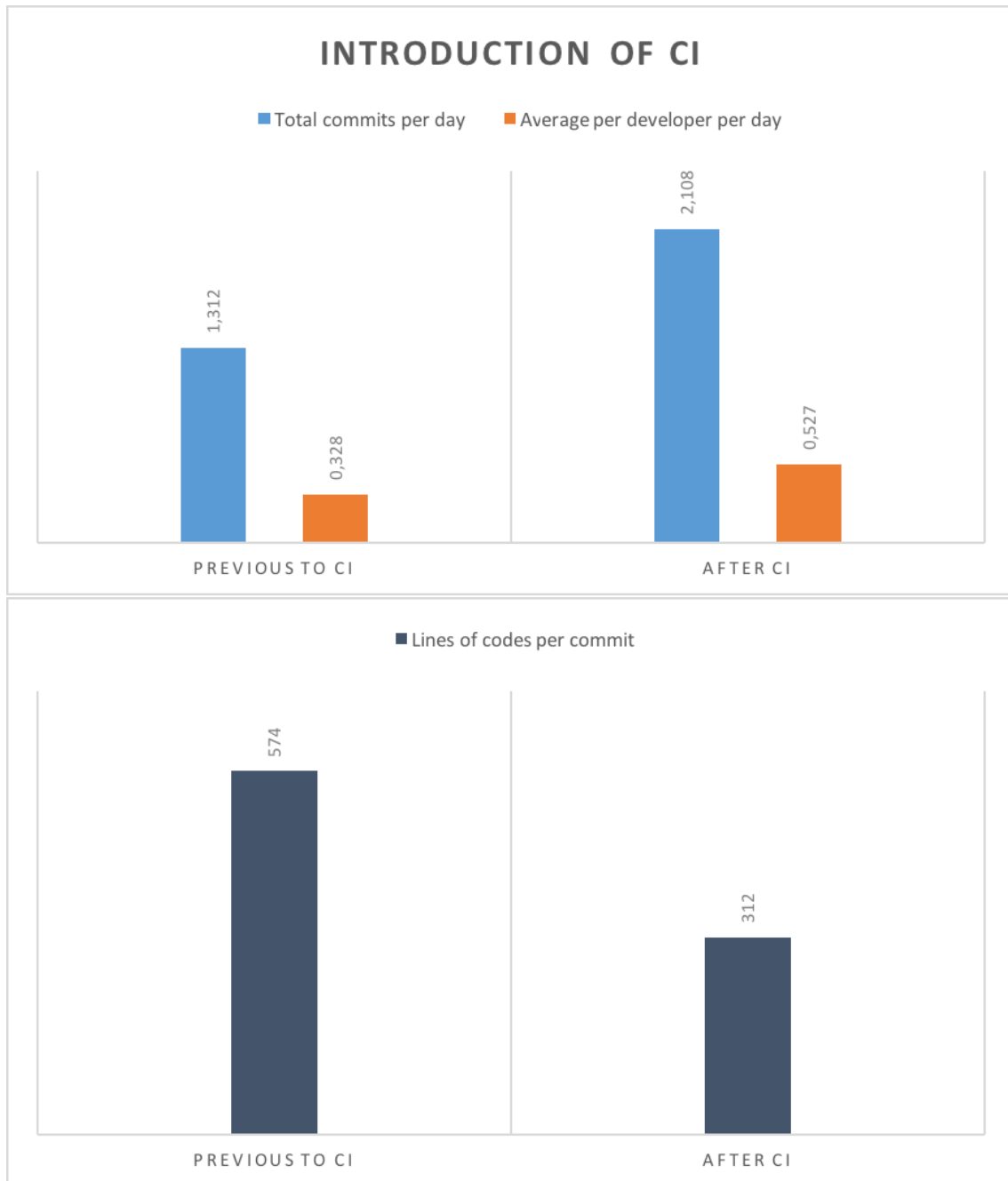


Figure 4.2: Statistical data collected when CI was introduced at the organization.

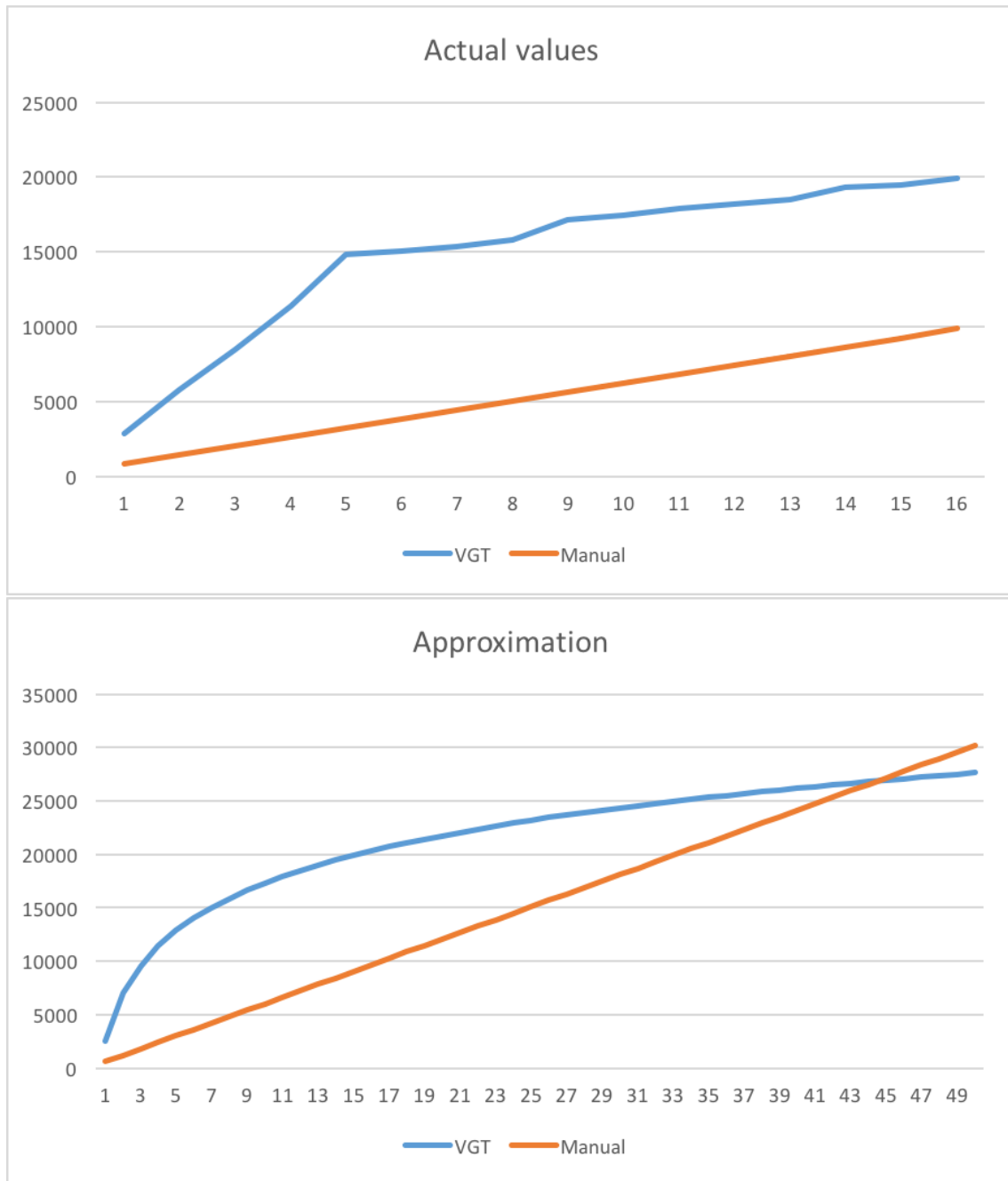


Figure 4.3: Actual and approximate descriptive values collected during the study. The y-axis shows the time spent in minutes and the x-axis shows the current week.

The actual values in figure 4.3 describe the cost in time to engineer the test suite in VGT. The result is based on engineering and refactoring of the initial test suite for the first five weeks. Afterwards the suite needs to be maintained and updated when new features are added. The second line figure 4.3 describes the time spent performing manual tests before the automated test suite was introduced. This data is an approximation made by the authors based on previous domain knowledge in collaboration with interviews made with the development team.

The approximation is the fitted model over how this data will grow over time. The result obtained shows that it takes 44 weeks before the developers will spend less time on testing with the help of VGT than they did with manual tests.

$$y = 7599.6 \ln(x) - 1013, 1 \quad (4.1)$$

Logarithmic approximation of how the cost of VGT will grow over time.

$$y = 607.35x - 753, 54 \quad (4.2)$$

Prioritization

To prioritize what tests to use under the time constraint in VGT, a formula that estimates how important a feature and the test belonging to the feature have been created by the authors. This was created during the third cycle which can be seen in figure 3.2. The formula will take four criterion into consideration and each feature will have a grade of 1-4 within each criteria, see appendix for full table. The reason the authors chose to prioritize the features and not each single test within the feature is because the test should give an indication if a feature is usable or not. This is because the stakeholders do not care as much about small bugs as to usability i.e. in their perspective all small bugs can not be fixed because it would cost too much to be acceptable. Furthermore, this formula has been created from the authors' experience of the SUT in combination with techniques from previous research, i.e. Test Case Prioritization (TCP) which identifies the efficient ordering of test cases to maximize certain properties [47].

Execution time - The time to execute a test is an important factor for a VGT test case, since it has longer execution time than other test techniques for automated tests. A long test shall not be executed too often if it does not cover a lot of vital functions.

Failure rate - If a test tends to fail often, it shall be executed more frequently. This is because there is a high possibility that the functions failing are affected by changes made to other parts of the system. A test is considered to be a failed test and placed in this category only when a root cause analysis can confirm that it is a real defect. This is to ensure that it is not a false positive or a badly written test that causes this, since those kinds of failures are irrelevant in this case.

Frequent usage - A feature or function that is used often by a stakeholder or user of the system, needs to be tested more frequently since they will have a bigger impact on the system if they fail in comparison to a feature used less often. The data to rank each feature have been taken from google analytics that keeps track of the site to see how frequent features are used.

System criticality - Features in a system have a different level of importance to the SUT. Features like making a purchase or logging in are mandatory for the company to generate profit from the customers, whilst a feature that changes the

image of a product is of importance, but there is no dependency issues if it suddenly stop working. Together with the stakeholders, the importance of each feature in the test suite will be graded.

From this, a formula combining all the factors have been made to get a value between 1-4 when all the factors are considered. After this, each test will be put into one of three categories, depending on the final priority number. Each category is clarified by a color scale; red, yellow and green. The number will indicate the overall significance of the test, and tell the testers which tests to include in the test suite. If the priority number is between 3-4 it will have the highest priority and a red color, if it is between 2-3 it will have the medium priority or a more yellow color and if it is between 1-2 it has the lowest priority or a more green color.

$$\begin{aligned} \textit{Execution time} &= E \\ \textit{Failure rate} &= Fr \\ \textit{Frequent usage} &= Fu \\ \textit{System criticality} &= S \\ \textit{Prioritization} &= \frac{2 * E + Fr + 2 * Fu + 2 * S}{7} \end{aligned} \tag{4.3}$$

Since VGT tests have a long execution time, there may not be enough time to run all the tests under the time constraint in CI. By assigning a number and color to each feature, there will be easier for the tester to include vital tests more often than non-vital tests. It is also important to note that a test cannot be left out in every run, since the whole SUT needs to be tested.

Tests in the red category needs to be executed in every test suite to ensure that the most vital parts of the SUT is always working. Tests in the yellow category are also of high importance, but not vital to the SUT. These tests needs to be executed weekly at least. Green is features that does not have too high impact on the SUT, and a test if this magnitude needs to be executed every two weeks, because these are features that does not rank high in multiple categories, and therefore are both expensive to execute and does not have a high impact on the SUT or high failure rate. The result from this formula can be seen in table 4.1 below.

4. Results

Test case	Priority	Test case	Priority
Admin features		User features	
Login	2,71	Login	3,29
Logout	2,71	Logout	2,43
CRM	2,14	Change account information	2,14
Private tickets	2,14	Make a purchase	3,71
Public tickets	2,14	Find products	3,29
FAQ	1,86	Change the shopping cart	3,29
Files	1,57	Register new user	2,14
Attributes (Manufacturer)	2,14	Verify errors in customer registration	2,43
Quotation	3,00	Update contact information	1,57
Add article	2,14	Change currency	1,86
Project	2,57	Tickets	1,57
Order	2,14	FAQ	1,86
Stock clearance	1,57	Subscribe to newsletter	1,86
Freights	1,29	Contact us	2,43
Campaign article	1,57	About	2,14
Remove article	1,86	Confirm changed content on site	2,43
Remove manufacturer	1,86	Search function	3,00
FTP	1,57	Terms and condition	1,86
Add user	1,86	Article check	3,00
Delete user	1,86		
Add admin	1,57		
Delete admin	1,86		
File category	2,43		
Orders	2,43		
Payment fees	2,14		
Exchange rate	1,86		
Admin view	2,14		
Article categories	1,86		
Matching categories	2,14		
News	2,14		
Banners	2,71		
Featurettes	2,71		
Hilights	2,43		
Partners	1,86		

Table 4.1: The result from the prioritization formula, the higher number (1-4)/the more red a feature is, means that the feature needs to be tested more often.

4.3 Evaluation Framework

The data obtained in this study have been mapped to the framework described in section 2.5 with the help of the data collection methods described in figure 3.5. The result can be seen in figure 4.4. Each square contains the change each cycle had with respect to BAPO as described in section 2.5 where the arrows shows how one change in a cycle carried over to the next cycle. If a square is white, no change could be measured in that area when the cycle had ended.

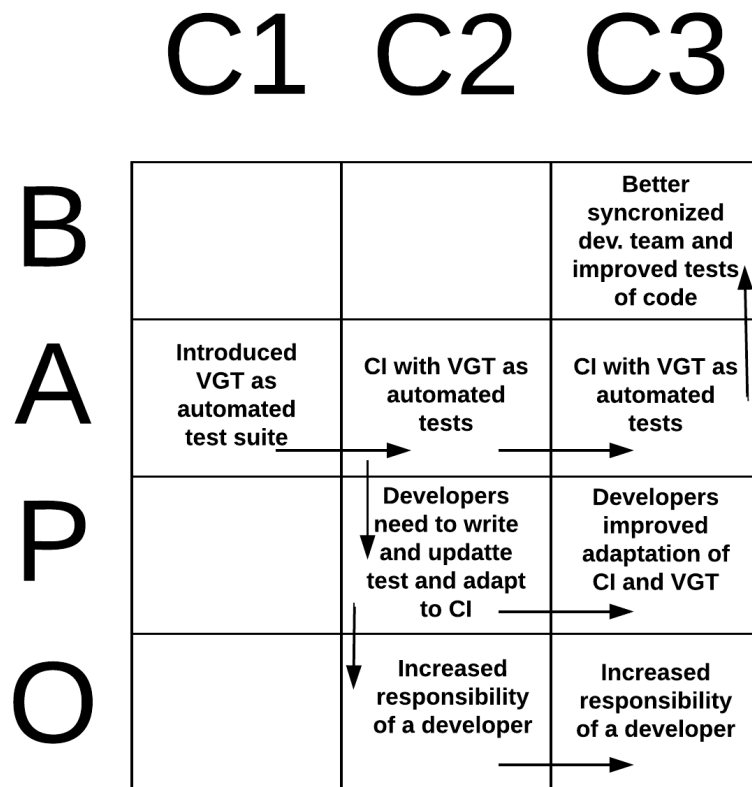


Figure 4.4: Mapping of the results to the chosen evaluation framework.

5

Discussion

This chapter will contain a discussion about the result obtained in this thesis. It will include limitations in this study, threats to validity and future work.

5.1 Implication of result

5.1.1 CI with VGT

As figure 4.2 shows, the introduction of CI led to an increase of commits to the main branch in the repository. This was expected, but the increase was not as high as assumed. Each developer should on average commit once a day when following the CI practice, according to Ståhl and Miller [3, 5]. The reason why we only got an average of one commit every second day can have various reasons, but our conclusion is that it takes time to adapt to the CI process, i.e. it is not easy to immediately know how to break down the task into small enough assignments so that it is always possible to finish it before the end of the working day. Furthermore, a working day for a developer is different from day to day, it is not always eight hours of coding, at least in our case. Some days a developer has other tasks, e.g. meetings, configurations, supplementary training, etc. All of these undertakings may also result in loss of commits per day, that should not be accounted for.

The reason that the total amount of lines of code per day has decreased from 753 to 658 lines during the introduction of CI can be many. Our perception is that due to the processes introduced, developers have tried to make smaller and more frequent commits. But it can also be that they have been more cautious when writing and committing code, as the theory of planned behavior predicts, people are more stringent when being studied [48]. The result of this has been that less code in total has been committed but the quality of the code has most likely increased. However, this is something that fades when time passes [48].

All developers in the team have a positive attitude towards CI and had some knowledge or previous experience of CI. However, several developers had a reluctant approach towards creating and maintaining tests. This implies that they accept the new processes that CI involve, as long as it does not affect their daily tasks too much and that they can understand the benefits of the processes. This is also confirmed by previous studies as one obstacle when implementing an agile process into an organization [49].

The results indicate that VGT works as automated tests in an CI environment, but the authors believe it needs to be complemented with tests on other abstraction

levels.

<i>Advantages</i>	<i>Disadvantages</i>	<i>Limitations</i>
Good for regression testing	VGT-tests have long execution time	Root cause analysis
Easy to create tests	Large upfront cost	Requires reference computer
No false negatives	Many false positives	
Indicates system performance		

Table 5.1: Found advantages, disadvantages and limitations during this study

The advantages, disadvantages and limitations can be seen in table 5.1, below is a list explaining each list item.

Advantages

Good for regression tests - A test suite in CI has the main purpose to adhere to the standard set by the stakeholder [5], to ensure this it is very beneficial to perform a regression test over the SUT and make sure that new committed code does not affect the functionality. Furthermore, it is a test technique that find all kinds of errors in the code-base and these two properties makes VGT a test technique that is very attractive to use in CI.

Easy to create and maintain tests - In CI, the test suite needs to be maintained and updated daily, since new code is committed often. It is easy to understand VGT and how to create and maintain the test suite. VGT is also a black box testing technique, so if needed, dedicated testers without knowledge of the code base could engineer and maintain the test suite to ensure that it is up to date. The tests written in VGT also has no direct references to the API of the SUT. This makes it possible for the developers to update and change the API, something that may occur frequently with all the commits in CI. The implication of this is that the tests written for specific functions does not need to be updated if the code is changed, but the functionality is kept the same. One example would be if the search algorithm is changed to optimize the runtime of the algorithm. If there was API changes, a test with references would need to be updated while a VGT test will not need any refactoring at all.

No false negatives - During this study the authors have not found any false negatives. This could be labeled a disadvantage as well, since there is a possibility that there are defects that have gone through the SUT. However, the daily usage of the SUT in combination with no reported errors at all from users since introducing VGT, gives us the implication that no errors have passed through the automated test suite. This renders a picture of a test suite that the testers can rely on and expect to catch the errors that occurs when engineering new software.

Indicates system performance - The setup allows VGT to test some parts of the SUT for system performance i.e. if the server has functions that have an increased

loading time. For example, if an array takes too long time to load or makes the GUI look abnormal, VGT will find this abnormality. In a CI context, this have the possibility to occur over time. A lot of small changes is made, and the constant updating of the software has a chance to backfire later on when a much larger data set is presented compared to when a test was created. As such, VGT can also support regression performance testing and perceivably testing on other quality attributes as well.

Disadvantages

VGT scripts have long execution time - A test written in VGT usually have a longer execution time than the same test using another kind of test technique. This is a disadvantage in a CI process, since it minimizes the different ways to implement CI and one of the main requirements with CI is rapid feedback time [17]. The running time is affected by computing power of the reference computer, the size of the screen of the reference computer as well as the size of the test suite. Furthermore, other test techniques can be parallelized easier than VGT tests, which otherwise would decrease the execution time of the tests.

Large upfront cost - The introduction of VGT in CI will have a large upfront investment, since at the very least VGT as a technique needs to be learned. If CI is new, this process needs to be taught as well. This upfront cost may stop companies from transitioning from their current processes. However, this study has shown that in the given context the upfront cost is returned over time by reducing time spent on manual testing and on merge conflicts.

False positives - There is a high percentage of false positives when using VGT. Since the test suite is executed at least daily in CI, this is a disadvantage that will be enlarged and will have the consequence that the developers needs to address the errors that is reported and fix them so the build is not classified as broken in CI. The amounts of false positives break the flow that CI aims to solve with a more streamlined development since the developers needs to perform a root cause analyses on every failed test.

Limitations

Root cause analysis - As stated previously VGT finds all kinds of errors, but it does not display what kind of error has occurred, or even if it is a real defect. In order to understand what kind of defect that the test have indicated, a root cause analysis needs to be performed. This is a task that can be both difficult and tedious and there is a risk that testers will get annoyed if they have to spend too much time on this task.

Requires a reference computer - VGT requires a reference computer, since different screens will show images differently. In the CI context, this means that tests for newly integrated code cannot be written on the developer's own computer. Neither can two developers write and update the SUT at the same time. This is a limitation since the test suite shall be updated after each developer is done with his or her code. There are several solutions to the issue with a reference computer, in this study we have only tested one, more about this can be found in 3.10.3.

The advantages is of such an high level that VGT shall not be removed from the test suite as a whole, but some tests might be better suited to test as unit tests, or complement the current tests with more lower level tests. The main disadvantages to get a test suite that is as close to optimal as possible is that the runtime is a bit long and that a root cause analysis needs to be made when a VGT test is returning an error. If one had unit tests on these parts as well, it would be easier to see if the code is faulty or if the error is caused during runtime.

The biggest question when introducing the high level tests in CI is the time constraint that CI runs under. If the test suite has too long runtime, there will be no time to fix the errors in the code before a new build needs to be started. In this particular domain we have shown that we are able to get a test suite that has a high enough test coverage and still fits in the time constraint. This was achieved by prioritizing the tests after factors to decide how often each test shall be included. This gives the testers an indication of which tests to pick when making a test suite containing a sub-set of the tests. The formula to prioritize the tests worked good in our domain, but might not be suitable for other applications. By using this prioritization, the aim is to increase the amount of times critical functions are tested under the time constraint. Some features might not require as much or frequent testing as other, so to include these tests will not be as beneficial as the more critical ones if one would need to remove some tests from the SUT to fit in the time constraint. The prioritization has also been made so that every test needs to be executed at least once in a given time frame depending on in which category the test have been placed. This lets the whole test suite execute in the longer time constraint needed, while returning feedback to the developers quick enough to make CI work.

There was not enough time to capture the data to see the long term effect over how the organisation as a whole will change after the introduction of CI with VGT as the automated test suite. With the knowledge of the companies levels before the study, how it changed during the study and previous studies we have mapped the result to the framework originally introduced by Börjesson and Feldt [37] described in section 2.5. The result can be seen in figure 5.1.

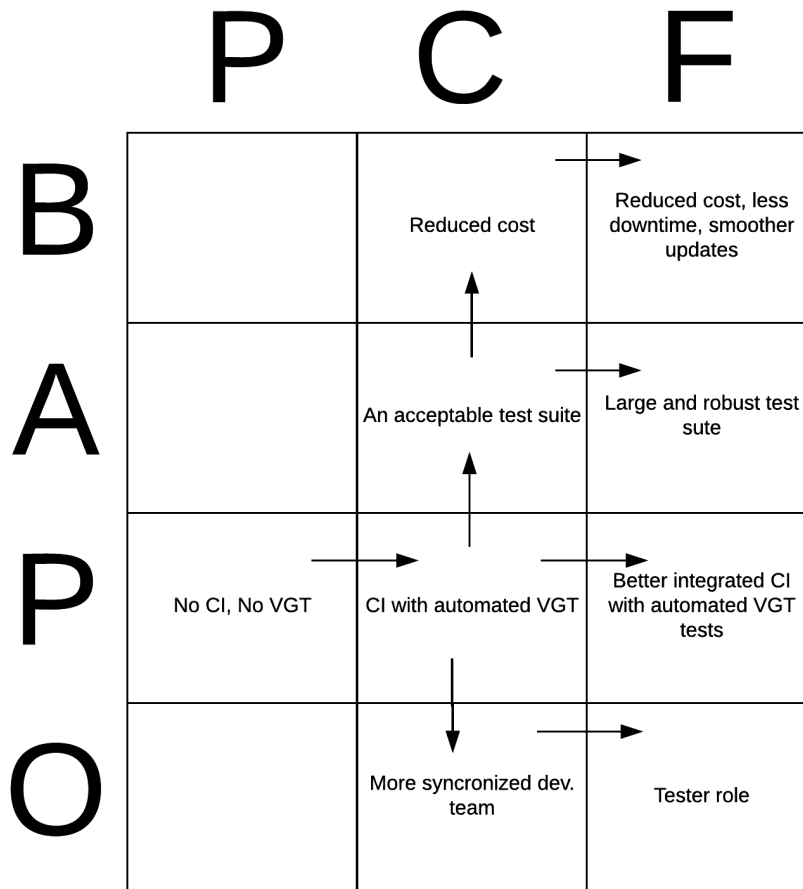


Figure 5.1: How the company have and possibly will change over time across multiple dimensions.

As one can see, all areas of a company is affected by the introduction, and they will continue to be affected as the process is even more adapted and refined to suite the context. There is no guarantee that the changes in the future column will happen, but the analysis made by the authors when comparing to previous studies shows that it is a likely scenario, at least in a general context. The test suite is bound to grow as new content is being added, and in the future a dedicated tester role will be needed to make sure that the test suite is not rotting and to make sure that there is a high enough test coverage [44]. The processes will also be better integrated and adapted when the developers have worked with it over time. The early result that it reduces costs by having smoother updates and more found defects early on indicates that the cost will be reduced even more over time.

5.1.2 Impact on the industry and the academia

This study shows that automated tests written in VGT has a place in CI industrial practice, but it may have to be modified from the way it was used in this study. The reason is that previous studies show that VGT works on multiple kinds of SUT's as long as it has a GUI, hence it is not limited to only the SUT used in this study[14]. However, it also shows that if a SUT is bigger than the one used in this study, a prioritization of tests will most likely be needed in order to avoid to long execution time of the test suite. CI as a practice is not only limited to this context, but rather a process which is widely generalizable. This implicates that both the techniques, VGT and the CI process can be used in other contexts and industries. For academia, the study shows that high level regression tests can be used as the only automated tests in a CI test suite. The result can be used to investigate if other parts of the Continuous family, Continuous Delivery where software shall be deliverable at all time, and Continuous Deployment where the software shall be deployed automatically after each build [15], can benefit from these high level tests as well. Furthermore, it shows the cost involved when introducing CI and VGT at a market competitive organization. This information can be used to see the values and limitations that exists when one or several theories is applied in a real world context and the reason behind the selection of theories.

5.1.3 Answers to research questions

All the above have helped us to answer our research questions, which can be seen in section 3.2:

RQ1

Firstly, the authors started by looking into sub-research question 1.1 in section 3.2. Here they looked at the advantages and disadvantages of VGT in a CI environment. In section 5.1 a table is provided with advantages, disadvantages and limitations found during this study, in the given context.

Secondly, sub-research question 1.2 in section 3.2 was investigated, what are the limitations with VGT as the test technique in CI. The limitations are also provided in the table provided in section 5.1.

Finally, the two sub-research questions where used to answer the first research question. The authors believe that the advantages overcome the disadvantages, one main reason for this is that VGT as a test technique is very suitable for regression testing. In the CI process good regression test is something that is very desirable. However, the authors still believe that a combination of VGT tests and low-level tests would be the best option as this would mitigate the limitations found in this study.

RQ2

In order to answer RQ2 the sub-research question 1 was answered first, how VGT test should be prioritized to maximize business value in CI. To answer this the

authors have created a formula that generated table 4.1. The table shows a color and number graded scale, which tells how to prioritize the features and test in the given context.

Then sub-research question 2 was answered, the advantages and disadvantages in business value when introducing VGT as the only test technique in CI. To answer this, the result was mapped to an evaluation framework, which can be seen in section 4.3. As in sub-research question 1.1, the advantages overcome the disadvantages. The costs and new processes needed to be implemented were relatively small, mainly due to the size of the organization and the advantages were in the long run a faster and smoother development process.

These two sub-research questions basically answer the second research question, by prioritizing the test according to the suggested formula and by knowing the startup cost when implementing VGT in CI. The business value will in the long run be improved and profitable according to the evaluation framework.

5.2 Limitations

In the middle of the study the company moved to new offices. This caused an interruption in the development process of test for more than a week. It is possible that this discontinuation slowed down the learning curve and the time it took to create all the initial tests, since it came at a critical time in the study when new test were still developed frequently.

Another limitation that occurred in the study is that the reference-computer had a disk crash. Backup existed on everything, but the process to replace the disk meant that no tests were executed for 72 hours. It is possible that this encouraged the developers into old habits, i.e. deploying code the way they did before CI was introduced at the company. Since no automated tests were running during the disk crash and thus prohibited the learning experience.

The company has a small team of developers, as team size is one of the factors that affect the ability to adapt to CI along with other factors such as project size and longevity [3]. So it might be more difficult to change the process and workflow of a larger organization with a larger development team that have used certain processes to engineer software for longer periods.

5.3 Threats to validity

5.3.1 Construct validity

Construct validity is defined to validate if the measurement has been made in the right place. i.e. do the data measure the relevant parts that the authors aim to measure. The study was conducted at a single company and in collaboration with one development team. The authors are normally part of the development team; this means that the authors had a personal interest. The data measured may have the potential to be positive biased since the authors have a positive attitude towards both CI and VGT. At the same time, the authors would want the implementation to

be a cost efficient process. If the transition ends up costing more than previously, the authors would in the end have invested more time for the same gain which would be counterproductive to the author's goal to organize and streamline the development process. As such, we perceive that the author's positive biases are countered by the author's need for VGT and CI to be implemented in a cost-efficient manner at the company. Also this is the first time the authors carry out a study of this magnitude, this means that even though literature in different areas have been followed, the lack of experience can still influence the result.

5.3.2 Internal validity

Internal validity is the concern that a factor that is not targeted in the measurement influence it, this can happen unwittingly. In this context, the factor that affects how the introduction is apprehended and how the quality of the code is changed, may be affected by other factors than the introduction of CI and VGT. Factors like the feeling of being observed or increased skill in coding over time are factors that can increase the quality of the code while the developments biased towards CI and VGT may interfere with the overall impression of CI with automated VGT tests. At the same time, the development team would like to have a streamlined process with high quality. If CI with VGT would not have achieved this, one would think that it would be risen to the authors attention to modify or stop the implementation. Furthermore, since the authors are acquainted with several of the interviewees, it may impinge the interviewees to give more positive feedback than they would otherwise because they want this study to be successful.

As the company neither had automated tests nor a CI process implemented before this study, some errors of deriving the actual cause of events may have occurred. That is, conclusions made by the authors may not be related to either CI or VGT, but a third unknown factor.

5.3.3 External validity

External validity is to what extent the result can be generalized in other contexts than were this study was performed. The result of this study is limited to one context only and with one development team. However, the result will give an indication on how the introduction of CI with automated VGT tests will look in other contexts. Neither CI nor VGT is limited to this context only, and have been applicable in other environments before. The result obtained in this study corresponds to earlier results in both areas, so it is most likely that the result obtained in the combination will be applicable in other contexts as well.

5.4 Reliability

Reliability is the aspects to what the result obtained in the study is dependent on the researchers conducting the research. The threat to reliability is that the authors had a very high domain knowledge compared to what is normally known when starting a study. Some knowledge may not be obtainable by doing a document

analysis or interviews alone. Furthermore, the authors were given authorization to select techniques and tools as long as it was inside the limits with respect to cost for the company. It is possible that the same deceptions will not be possible to implement in another context if the study is going to be replicated. The semi-structured interview may also be a reliability threat, since each question had the potential to raise a discussion, the same question asked by different interviewees and interview objects may result in a different kind of discussion for the same question.

5.5 Future work

This study has investigated VGT as the only automated tests in a CI process and has shown that VGT as the automated tests in CI have both its advantages and disadvantages. For future work, it would be of great value if a research could show that high level tests in combination with low-level test would complement the result provided in this thesis. The result in this study and previously conducted studies indicates that a combination will have the advantages of both techniques and mitigate the disadvantages of the techniques.

6

Conclusion

Continuous Integration (CI) is a process that is being more and more recognized in the software industry. To ensure high quality of the code-base, an automated test suite is required and in the industry these tests are usually low level tests, e.g. unit tests. Firstly, this study aims to see if it is possible to replace these commonly used low level tests with high level tests called Visual GUI Testing (VGT). Secondly, it reviews the advantages and disadvantages when introducing VGT as the only automated tests on a small web-development project. Finally, this study also investigates how a small organization is affected with regards to the BAPO (Business, Architecture, Process and Organization) framework, when introducing VGT as the only automated tests and what can be done in order to maximize the business value. It is of importance to show how applicable VGT is as the automated tests in CI because to the authors best knowledge, this has never been reviewed before. Furthermore, it is important to investigate how a profit-driven organization is affected when introducing the established CI process together with a test technique that is not as established in the market space to see if it is profitable.

The main contributions in this thesis is a statement on how applicable VGT as the only test technique in the CI process and how to mitigate the limitations in the given context. Furthermore, a statement over how to maximize the business value of an organization when transiting from no automated tests into a CI environment with VGT as the automated test technique in a small web-development project. The study also reflects how the cycles for the developers and the time spent on testing have been affected, along with the associated costs for these.

Bibliography

- [1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” 2002.
- [2] F. Cannizzo, R. Clutton, and R. Ramesh, “Pushing the boundaries of testing and continuous integration,” in *Agile, 2008. AGILE’08. Conference*, pp. 501–505, IEEE, 2008.
- [3] D. Ståhl and J. Bosch, “Modeling continuous integration practice differences in industry software development,” *Journal of Systems and Software*, vol. 87, pp. 48–59, 2014.
- [4] M. Meyer, “Continuous integration and its tools,” *Software, IEEE*, vol. 31, no. 3, pp. 14–16, 2014.
- [5] A. Miller, “A hundred days of continuous integration,” in *Agile, 2008. AGILE’08. Conference*, pp. 289–293, IEEE, 2008.
- [6] M. Olan, “Unit testing: test early, test often,” *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 319–328, 2003.
- [7] R. Miller and C. T. Collins, “Acceptance testing,” *Proc. XPUniverse*, vol. 238, 2001.
- [8] M. N. F. Emil Alégroth and H. H. Olsson, “Jautomate: a tool for system- and acceptance-test automation,” 2013.
- [9] A. Adamoli, D. Zapanu, M. Jovic, and M. Hauswirth, “Automated gui performance testing,” *Software Quality Journal*, vol. 19, no. 4, pp. 801–839, 2011.
- [10] W.-K. Chen, T.-H. Tsai, and H.-H. Chao, “Integration of specification-based and cr-based approaches for gui testing,” in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 1, pp. 967–972, IEEE, 2005.
- [11] S. Berner, R. Weber, and R. K. Keller, “Observations and lessons learned from automated testing,” in *Proceedings of the 27th international conference on Software engineering*, pp. 571–579, ACM, 2005.
- [12] E. Horowitz and Z. Singhera, “Graphical user interface testing,” *Technical report Us C-C S-93-5*, vol. 4, no. 8, 1993.
- [13] E. Börjesson and R. Feldt, “Automated system testing using visual gui testing tools: A comparative study in industry,” 2012.
- [14] R. F. Emil Alégroth and H. H. Olsson, “Transitioning manual system test suites to automated testing: An industrial case study,” 2013.
- [15] H. H. Olsson and J. Bosch, *Continuous Software Engineering*, ch. Climbing the “Stairway to Heaven”: Evolving From Agile Development to Continuous

- Deployment of Software, pp. 15–27. Cham: Springer International Publishing, 2014.
- [16] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.
- [17] M. Fowler and M. Foemmel, “Continuous integration,” *Thought-Works*) [http://www.thoughtworks.com/Continuous Integration. pdf](http://www.thoughtworks.com/Continuous%20Integration.pdf), p. 122, 2006.
- [18] J. Holck and N. Jørgensen, “Continuous integration and quality assurance: A case study of two open source projects,” *Australasian Journal of Information Systems*, vol. 11, no. 1, 2003.
- [19] A. Debbiche, M. Dienér, and R. Berntsson Svensson, *Product-Focused Software Process Improvement: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings*, ch. Challenges When Adopting Continuous Integration: A Case Study, pp. 17–32. Cham: Springer International Publishing, 2014.
- [20] R. Patton, *Software testing*, vol. 2. Sams Indianapolis, 2001.
- [21] P. J. Schroeder and B. Korel, *Black-box test reduction using input-output analysis*, vol. 25. ACM, 2000.
- [22] T. Xie, K. Taneja, S. Kale, and D. Marinov, “Towards a framework for differential unit testing of object-oriented programs,” in *Proceedings of the Second International Workshop on Automation of Software Test*, p. 5, IEEE Computer Society, 2007.
- [23] P. C. Jorgensen and C. Erickson, “Object-oriented integration testing,” *Communications of the ACM*, vol. 37, no. 9, pp. 30–38, 1994.
- [24] H. K. Leung and L. White, “A study of integration testing and software regression at the integration level,” in *Software Maintenance, 1990, Proceedings, Conference on*, pp. 290–301, IEEE, 1990.
- [25] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991.
- [26] R. Black, *Managing the testing process*. John Wiley & Sons, 2002.
- [27] E. Alégroth, “Random visual gui testing: Proof of concept,” 2013.
- [28] I. Jovanović, “Software testing methods and techniques,” *The IPSI BgD Transactions on Internet Research*, vol. 30, 2006.
- [29] E. Alégroth, Z. Gao, R. A. Oliveira, and A. Memon, “Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study,” in *The Proceedings of eighth edition of the IEEE International Conference on Software Testing, Verification, and Validation (ICST 2015)*, 2015.
- [30] A. Bruns, A. Kornstädt, and D. Wichmann, “Web application tests with selenium,” *Software, IEEE*, vol. 26, no. 5, pp. 88–91, 2009.
- [31] I. X. S. (e-book collection), *Software and systems engineering Software testing Part 2: Test processes*. S.l.: s.n., 2013.
- [32] J. Palat, “Introducing vagrant,” *Linux Journal*, vol. 2012, no. 220, p. 2, 2012.
- [33] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

- [34] A. Collins, D. Joseph, and K. Bielaczyc, "Design research: Theoretical and methodological issues," *The Journal of the learning sciences*, vol. 13, no. 1, pp. 15–42, 2004.
- [35] A.-L. Fayard and J. Weeks, "Photocopiers and water-coolers: The affordances of informal interaction," *Organization studies*, vol. 28, no. 5, pp. 605–634, 2007.
- [36] S. Sarker, S. Sarker, D. Nicholson, and K. Joshi, "Knowledge transfer in virtual information systems development teams: an empirical examination of key enablers," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pp. 10–pp, IEEE, 2003.
- [37] E. Börjesson and R. Feldt, "Structuring software engineering case studies to cover multiple perspectives," in *SEKE*, pp. 276–281, 2011.
- [38] G. A. Bowen, "Document analysis as a qualitative research method," *Qualitative research journal*, vol. 9, no. 2, pp. 27–40, 2009.
- [39] F. van der Linden, "Family evaluation framework-overview & introduction," *Philips Medical Systems, version*, vol. 1, 2005.
- [40] A. Deshpande and D. Riehle, "Continuous integration in open source software development," in *IFIP International Conference on Open Source Systems*, pp. 273–280, Springer, 2008.
- [41] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on visual gui testing," *Information and Software Technology*, vol. 73, pp. 66–80, 2016.
- [42] G. Goldkuhl, "Action research vs. design research: using practice research as a lens for comparison and integration," in *The 2nd international SIG Prag workshop on IT Artefact Design & Workpractice Improvement (ADWI-2013), 5 June, 2013, Tilburg, the Netherlands*, 2013.
- [43] J. Iivari and J. Venable, "Action research and design science research—seemingly similar but decisively dissimilar," in *European Conference on Information Systems*, vol. 17, pp. 1–13, 2009.
- [44] E. Alégroth, M. Steiner, and A. Martini, "Exploring the presence of technical debt in industrial gui-based testware: A case study," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 257–262, IEEE, 2016.
- [45] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *Software metrics, 2005. 11th IEEE international symposium*, pp. 10–pp, IEEE, 2005.
- [46] "CAPTCHA: telling humans and computers apart automatically." <http://www.captcha.net/>. Accessed: 2016-07-12.
- [47] C. Catal and D. Mishra, "Test case prioritization: a systematic mapping study," *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2013.
- [48] I. Ajzen, "The theory of planned behavior," *Organizational behavior and human decision processes*, vol. 50, no. 2, pp. 179–211, 1991.
- [49] B. Boehm and R. Turner, "Management challenges to implementing agile processes in traditional development organizations," *Software, IEEE*, vol. 22, no. 5, pp. 30–39, 2005.

A

Appendix 1

Test case	System criticality	Frequency	Execution Time	Failures
User features				
Login	4	3	4	1
Logout	2	2	4	1
Change account information	2	2	3	1
Make a purchase	4	4	3	4
Find products	4	4	3	1
Change the shopping cart	4	4	3	1
Register new user	4	1	2	1
Verify errors in customer registration	3	1	4	1
Update contact information	2	1	2	1
Change currency	1	1	4	1
Tickets	2	1	2	1
FAQ	1	1	4	1
Subscribe to newsletter	1	1	4	1
Contact us	2	2	4	1
About	1	2	4	1
Confirm changed content on site	1	4	3	1
Search function	3	4	3	1
Terms and condition	1	1	4	1
Article check	2	4	4	1
Test case	System criticality	Frequency	Execution Time	Failures
Admin features				
Login	4	1	4	1
Logout	2	3	4	1
CRM	4	2	1	1
Private tickets	3	1	3	1
Public tickets	3	1	2	3
FAQ	1	3	2	1
Files	2	1	2	1
Attributes (Manufacturer)	1	2	4	1
Quotation	3	4	3	1
Add article	3	3	1	1
Project	3	4	1	2
Order	4	1	2	1
Stock clearance	2	1	2	1
Freights	2	1	1	1
Campaign article	1	1	3	1
Remove article	1	1	4	1
Remove manufacturer	1	1	4	1
FTP	1	1	3	1
Add user	3	1	2	1
Delete user	1	1	4	1
Add admin	2	1	2	1
Delete admin	1	1	4	1

A. Appendix 1

Test case	System criticality	Frequency	Execution Time	Failures
Admin features				
File category	1	4	3	1
Orders	4	2	2	1
Payment fees	2	1	4	1
Exchange rate	2	1	3	1
Admin view	2	1	4	1
Article categories	2	1	3	1
Matching categories	1	3	3	1
News	2	3	2	1
Banners	3	3	3	1
Featurettes	3	3	3	1
Hilights	3	1	4	1
Partners	1	1	4	1

Table A.1: Valuation of the test cases in the four categories from section 4.2