Thesis for The Degree of Doctor of Philosophy

# Visual GUI Testing:
## Automating High-Level Software Testing in Industrial Practice

Emil Alégroth


Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden, 2015

**Visual GUI Testing:**
**Automating High-Level Software Testing in Industrial Practice**

Emil Alégroth



Technical Report No 117D
ISSN 0346-718X
ISBN 978-91-7597-227-5
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Göteborg, Sweden 2015.

*To Therese, Alexandra and my supporting family*

# Abstract

Software Engineering is at the verge of a new era where continuous releases are becoming more common than planned long-term projects. In this context test automation will become essential on all levels of system abstraction to meet the market's demands on time-to-market and quality. Hence, automated tests are required from low-level software components, tested with unit tests, up to the pictorial graphical user interface (GUI), tested with user emulated system and acceptance tests. Thus far, research has provided industry with a plethora of automation solutions for lower level testing but GUI level testing is still primarily a manual, and therefore costly and tedious, activity in practice.

We have identified three generations of automated GUI-based testing. The first ($1^{st}$) generation relies on GUI coordinates but is not used in practice due to unfeasible maintenance costs caused by fragility to GUI change. Second ($2^{nd}$) generation tools instead operate against the system's GUI architecture, libraries or application programming interfaces. Whilst this approach is successfully used in practice, it does not verify the GUI's appearance and it is restricted to specific GUI technologies, programming languages and platforms.

The third ($3^{rd}$) generation, referred to as Visual GUI Testing (VGT), is an emerging technique in industrial practice with properties that mitigate the challenges experienced with previous techniques. VGT is defined as a tool-driven test technique where image recognition is used to interact with, and assert, a system's behavior through its pictorial GUI as it is shown to the user in user-emulated, automated, system or acceptance tests. Automated tests that produce results of quality on par with a human tester and is therefore an effective complement to reduce the aforementioned challenges with manual testing. However, despite its benefits, the technique is only sparsely used in industry and the academic body of knowledge contains little empirical support for the technique's industrial viability.

This thesis presents a broad evaluation of VGT's capabilities, obtained through a series of case studies and experiments performed in academia and Swedish industry. The research follows an incremental methodology that began with experimentation with VGT, followed by industrial studies that were concluded with a study of VGT's use at a company over several years. Results of the research show that VGT is viable for use in industrial practice with better defect-finding ability than manual tests, ability to test any GUI based system, high learnability, feasible maintenance costs and both short and long-term company benefits. However, there are still challenges associated with the successful adoption, use and long-term use of VGT in a company, the most crucial that suitable development and maintenance practices are used. This thesis thereby concludes that VGT can be used in industrial practice and aims to provides guidance to practitioners that seek to do so. Additionally, this work aims to be a stepping stone for academia to explore new test solutions that build on image recognition technology to improve the state-of-art.



### Keywords

Software Engineering, Automated Testing, Visual GUI Testing, Industrial Research, Empirical Research, Applicability and Feasibility

# Acknowledgments

First and foremost, my deepest thanks go to my main supervisor, friend and mentor Professor Robert Feldt whose belief in me and unwavering support made this thesis possible. We have had an amazing journey together and you have not just taught me how to be a researcher but a better person as well, something that I will cherish forever.

Second, my thanks go to my second supervisor, Associate professor Helena Holmström-Olsson, whose positive attitude, support and advice have been a great source of inspiration and help, both in times of joy and despair.

Next I want to thank my examiner Professor Gerardo Scheider and all my past and present colleagues at the Software Engineering division at Chalmers University of Technology whose guidance and support has been invaluable for the completion of my thesis work. In particular I would like to thank Dr. Ana Magazinius, Dr. Ali Shahrokni, Dr. Joakim Pernstål, Pariya Kashfi, Antonio Martini, Per Lenberg, Associate professor Richard Berntsson Svensson, Professor Richard Torkar and Professor Jan Bosch for many great experiences but also for always being there to listen to and support my sometimes crazy ideas. Additionally, I want to thank Bogdan Marculescu and Professor Tony Gorschek who, together with Robert, convinced me, in their own way, to proceed a PhD. Further, I want to thank my international research collaborators, in particular Professor Atif Memon, Rafael Oliveira and Zebao Gao who made a research visit in the US a wonderful experience.

However, this thesis had not been completed without the support of my loving wife, and mother of my wonderful Alexandra, Therese Alégroth. She has been my rock and the person I could always rely on when times were tough. Thanks also go to my mother Anette, father Tomas and sister Mathilda for believing in me and for their sacrifices to ensure that I could pursue this dream. Further, I want to thank my friends for always being there and I hope that one day, perhaps after reading my thesis, that you will understand what I do for a living.

I also want to thank my industrial collaborators, in particular the staff at Saab AB, Michel Nass, the staff at Inceptive, Geoffrey Bache, the Software Center and everyone else that has helped, supported and believed in my research.

This research has been conducted in a joint research project financed by the Swedish Governmental Agency of Innovation Systems (Vinnova), Chalmers University of Technology and Saab AB. My studies were also supported by the Swedish National Research School for Verification and Validation (SWELL), funded by Vinnova.

# List of Publications

## Appended papers

This thesis is primarily supported by the following papers:

1. E. Börjesson, R. Feldt, "Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry"
   *Proceedings of the 5th International Conference on Software Testing Verification and Validation (ICST'2012), Montreal, Canada, April 17-21, 2012* pp. 350-359.

2. E. Alégroth, R. Feldt, H. H. Olsson, "Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study"
   *Proceedings of the 6th International Conference on Software Testing Verification and Validation (ICST'2013), Luxenbourg, March 18-22, 2013.*

3. E. Alégroth, R. Feldt, L. Ryrholm, "Visual GUI Testing in Practice: Challenges, Problems and Limitations"
   *Published in the Empirical Software Engineering Journal, 2014.*

4. E. Alégroth, R. Feldt, P. Kolström, "Maintenance of Automated Test Suites in Industry: An Empirical study on Visual GUI Testing"
   *In submission.*

5. E. Alégroth, R. Feldt, "On the Long-term Use of Visual GUI Testing in Industrial Practice: A Case Study"
   *In submission.*

6. E. Alégroth, G. Zebao, R. Oliviera, A. Memon, "Conceptualization and Evaluation of Component-based Testing Unified with Visual GUI Testing: An Empirical Study"
   *Proceedings of the 8th International Conference on Software Testing Verification and Validation (ICST'2015), Graz, Austria, April 13-17, 2015*

7. E. Alégroth, J. Gustafsson, H. Ivarsson, R. Feldt, "Replicating Rare Software Failures with Visual GUI Testing: An Industrial Success Story"
   *Accepted for publication in the Journal of IEEE Software, 2015.*

## Other papers

The following papers are published but not appended to this thesis, either due to overlapping contents to the appended papers, contents not related to the thesis or because the contents are of less priority for the thesis main conclusions.

1. E. Börjesson, R. Feldt, "Structuring Software Engineering Case Studies to Cover Multiple Perspectives"
   *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Miami Beach, Florida, USA, July 1-3, 2011.*

2. E. Alégroth, M. Nass, H. H. Olsson, "JAutomate: a Tool for System- and Acceptance-test Automation"
   *Proceedings of the 6th International Conference on Software Testing, Verification and Validation (ICST'2013), Luxenbourg, March 18-22, 2013.*

3. E. Alégroth, "Random Visual GUI Testing: Proof of Concept"
   *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2013), Boston, Massachusetts, USA, June 27-29, 2013.*

4. G. Liebel, E. Algroth and R.Feldt, "State-of-Practice in GUI-based System and Acceptance Testing: An Industrial Multiple-Case Study"
   *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2013.*

5. E. Algroth and R.Feldt, "Industrial Application of Visual GUI Testing: Lessons Learned"
   *Chapter of the book Continuous Software Engineering published by Springer, 2014.*

6. E. Alégroth, G. Bache, E. Bache, "On the Industrial Applicability of TextTest: An Empirical Case Study"
   *Proceedings of the 8th International Conference on Software Testing Verification and Validation (ICST'2015), Graz, April 13-17, 2015*

7. R. Oliviera, E. Alégroth, G. Zebao, A. Memon, "Definition and Evaluation of Mutation Operators for GUI-level Mutation Analysis"
   *Proceedings of the 10th Mutation Workshop (Mutation'2015), Graz, Austria, April 13, 2015*

## Statement of contribution

In all listed papers, the first author was the primary contributor to the research idea, design, data collection, analysis and/or reporting of the research work.

# Contents

# Chapter 1

# Introduction

## 1.1 Introduction

Today, software is ubiquitous in all types of user products, from software applications to cars, mobile applications, medical systems, etc. Software allows development organizations to broaden the number of features in their products, improve the quality of these features and provide customers with post-deployment updates and improvements. In addition, software has shortened the time-to-market in many product domains, a trend driven by the market need for new products, features and higher quality software.

However, these trends place new time constraints on software development organizations that limit the amount of requirements engineering, development and testing that can be performed on new software [1]. For testing, these time constraints imply that developers can no longer verify and validate the software's quality with manual test practices since manual testing is associated with properties such as high cost, tediousness and therefore error-proneness [2–7]. These properties are a particular challenge in the context of changing requirements where the tests continuously need to be rerun for regression testing [8, 9].

Automated testing has been suggested as the solution to this challenge since automation allows tests to be run more frequently and at lower cost [4, 7, 10]. However, most automated test techniques have prerequisites that prohibit their use on software written in certain programming languages, for certain operating systems, platforms, etc. [4, 11–13]. Additionally, most automated test techniques operate on a lower level of system abstraction, i.e. against the backend of the system. One such, commonly used, low-level test technique is automated unit testing [14]. Whilst unit tests are applicable to find defects in individual software components, its use for system and acceptance testing is still a subject of ongoing debate [15, 16]. Test techniques exist for automated system and acceptance testing that interact with the system under test (SUT) through hooks into the SUT or its GUI. However, these techniques do not verify that the pictorial GUI, as shown to the user, behaves or appears correctly. These techniques therefore have limited ability to fully automate manual, scenario-based, regression test cases, in the continuation of this thesis referred to as *manual test cases*. Consequently, industry is in need of a

flexible and GUI-based test automation technique that can emulate human tester behavior to mitigate the challenges associated with current manual and automated test techniques.

In this thesis we introduce and evaluate Visual GUI Testing (VGT). VGT is a term we have defined that encapsulates all tools that use image recognition to interact with a SUT's functionality through the bitmaps shown on the SUT's pictorial GUI. These interactions are performed with user emulated keyboard and mouse events that make VGT applicable on almost any GUI-driven application and to automate test cases that previously had to be performed manually. Consequently, VGT has the properties that software industry is looking for in a flexible, GUI-based, automated test technique since the technique's only prerequisite is that a SUT has a GUI. A prerequisite that only limits the technique's applicability and usefulness for, as examples, server or other backend software.

However, at the start of this thesis work the body of knowledge on VGT was limited to analytical research results [17] regarding VGT tools, i.e. Triggers [18], VisMap [19] and Sikuli [20]. Hence, no empirical evidence existed regarding the technique's applicability or feasibility of use in industrial practice. Applicability that, in this thesis, refers to factors such as a test technique's defect-finding ability, usability for regression, system and acceptance testing, learnability and flexibility of use for different types of GUI-based software. Feasibility, in turn, refers to the long-term applicability of a technique, including feasible development and maintenance costs, usability under strict time constraints and suitable time until the technique provides positive return on investment (ROI). Empirical evidence on these factors are key to understand the real life complexities of using the technique, to build best practices and to advance its use in industrial practice [17, 21]. However, such evidence can only be acquired through an incremental process that evaluates the technique from several perspectives and different industrial contexts. This thesis work was therefore performed in Swedish software industry, with different projects, VGT tools and research techniques to fulfill the thesis research objective. Hence, to acquire evidence for, or against, the applicability and feasibility of adoption, use and viability of VGT in industrial practice, including what challenges, problems and limitations that are associated with these activities. Work that consequently resulted in an overall understanding of the current state-of-practice of VGT, what impedes its continued adoption and a final, yet positive, conclusion regarding the long-term viability of VGT in industrial use.

The results presented in this introductory chapter (Chapter 1) are structured as follows. First, an introduction is given in Section 1.1 followed by a background to this research, including; manual, automated and automated GUI-based testing. Section 1.3 then presents the research problem, questions and the methodology. This section also details the different research methods that were used and how the included papers contribute to answer the thesis research questions. An overview, and summaries, of the included papers are then given in Section 1.4. Section 1.5 then presents the syntheses of included papers and finally the thesis introduction is concluded in a summary in Section 1.6.

## 1.2 Software engineering and the need for testing

Software engineering is the application of engineering best practices in a structured process to design, develop and maintain software of high quality [22]. Several software development processes have been defined such as plan-driven, incremental and agile development processes [23, 24]. These processes can be divided into three fundamental activities: requirements engineering, development (design and implementation) and verification and validation.

Requirements engineering refers to the activity of elicitation, specification and modeling of the software's requirements, i.e. the needs of the customer/user. Hence, features, functions and qualities that the developed software must include [25, 26]. In turn, development is the activity of designing and realizing the requirements in software that fulfills the user's needs. Finally, verification and validation, traditionally, is the activity of evaluating that the developed software conforms to the requirements [1], most commonly achieved through testing.

Tests for verification and validation are therefore a tightly coupled counterpart to requirements [27]. Hence, whilst the quality of a software system is determined by how well each process activity is performed, it is through testing that this quality is measured. Measurements that can be taken throughout the development process, i.e. early with reviews of documents or code or late with customer acceptance tests. Testing is therefore an essential activity in all software engineering, regardless of process or development objective.

### 1.2.1 Software Testing

Software testing for verification and validation is a core, but also costly, activity that can make up for 20-50 percent of the cost of a software development project [1, 28, 29]. Verification is defined as the practice of assuring that the SUT conforms to its requirements, whilst validation is defined as the practice of assuring that the SUT conforms to the requirements and fulfills the user's needs [25, 26].

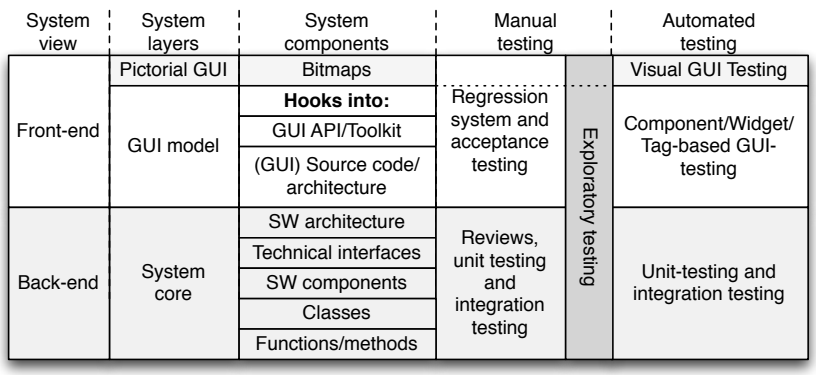


Figure 1.1: *Theoretical, layered, model of a System and the manual/automated techniques generally used to test the different layers.*

Testing for the purpose of verification can be split into three types; unit, integration and system testing [30], which are performed on different levels of system abstraction [16, 26, 31] as shown in Figure 1.1. A unit test verifies that the behavior of a single software component conforms to its low-level functional requirement(s) and is performed either through code reviews or more commonly through automated unit tests [9, 11, 14, 15, 32–34]. In turn, integration tests verify the conformance of several components' interoperability between each other and across layers of the SUT's implementation [16, 30]. Components can in this context be single methods or classes but also hardware components in embedded systems. Finally, system tests are, usually, scenario-based manual or automated tests that are performed either against the SUT's technical interfaces or the SUT's GUI to verify that the SUT, as a whole [30], conforms to its feature requirements [35–37]. However, scenario-based tests are also used to validate the conformance of a SUT in acceptance tests that are performed either by, or with, the SUT's user or customer [35–38]. The key difference between system and acceptance test scenarios is therefore how representative they are of the SUT's real-world use, i.e. the amount of domain knowledge that is embedded in the test scenario.

Testing is also used to verify that a SUT's behavior still conforms to the requirements after changes to the SUT, i.e. regression tests. Regression tests can be performed with unit, integration, system or acceptance test cases that have predefined inputs for which there are known, expected, outputs [9]. Inputs and outputs that are used to stimulate and assert various states of the SUT. As such, the efficiency of a regression test suite is determined by the tests' coverage of the SUT's components, features, functions, etc [34, 39], i.e. the amount of a SUT's states that are stimulated during test execution. This also limits regression tests to finding defects in states that are explicitly asserted, which implies that the test coverage should be as high as possible. However, for manual regression tests, high coverage is costly, tedious and error-prone [2–7], which is the primary motivation why automated testing is needed and should be used on as many different levels of system abstraction as possible [16, 40]. Especially in the current market where the time available for testing is shrinking due to the demands for faster software delivery [1]. Demands that have transformed automated testing from "want" to a "must" in most domains.

However, whilst lower levels of system abstraction are well supported by automated regression test techniques, tools and frameworks, there is a lack of automated techniques for testing through the pictorial GUI, i.e. the highest level of system abstraction. Thus, a lack of support that presents the key motivator for the research presented in this thesis.

To cover any lack of regression test coverage, exploratory testing, defined as simultaneous learning, test design and test execution, is commonly used in industrial practice [41, 42]. The output of exploratory testing is a defect but also the scenario(s) that caused the defect to manifest, i.e. scenarios that can be turned into new regression tests. This technique has been found to be effective [43] but has also been criticized for not being systematic enough for fault replication. Further, the practice requires decision making to guide the testing and is therefore primarily performed manually, despite the existence of a few automated exploratory testing tools, e.g. CrawlMan [44]. However, automated exploratory testing is still an unexplored research area that war-

rants more research, including automated GUI-based exploratory testing since it could help mitigate the challenges associated with manual verification and validation, e.g. cost.

In summary, testing is used in industrial practice on different levels of system abstraction for verification and validation of a SUT's conformance to its requirements. However, much of this testing is manual, which is costly, tedious and error prone, especially for manual regression testing, which is suggested as solvable with automated testing. More research is therefore warranted into new automated test techniques and in particular techniques that operate against the SUT's highest level of system abstraction, i.e. the pictorial GUI.

### 1.2.2 Automated Software Testing

There are two key motivators for the use of automated testing in industrial practice; (1) to improve software quality and (2) to lower test related costs [40].

***Software quality*:** Automated tests help raise software quality through higher execution speed than manual tests that allow them to be executed more frequently [16, 40]. Higher test frequency provides faster feedback to the developers regarding the quality of the software and enables defects to be caught and resolved earlier. In turn, quick defect resolution lowers the project's development time and mitigates the chance of defect propagation into customer deliveries. Early defect detection also mitigates synergy effects to occur between defects, for instance that two or more defects cause a joint failure which root-cause therefore becomes more difficult and costly to find.

However, a prerequisite for any automated test technique to be used frequently is that the tests have reasonable test execution time. This prerequisite is particularly important in contexts where the tests are used for continuous integration, development and deployment [45]. Hence, contexts where the test suites should be executed each time new code is integrated to the SUT, e.g. on commit, which cause the tests to set the pace for the highest possible frequency of integration. This pacing is one reason why automated unit tests [9, 11, 14, 15, 32–34] are popular in industrial practice since several hundred unit tests can be executed in a matter of minutes. In addition, unit tests are popular in agile software development companies, where they are used to counteract regression defects [46] caused by change or refactoring that is promoted by the process [47, 48].

***Lower cost*:** Automated testing is also used to lower the costs of testing by automating tests, or parts of tests, that are otherwise performed manually. However, there are still several costs associated with automated tests that need to be considered.

***First***, all automated test techniques require some type of tool that either needs to be acquired, bought and/or developed. Next, the intended users of the tool need be given training or time to acquire knowledge and experience with the tool and its technique before it can be used. Knowledge and experience that might be more or less cumbersome to acquire dependent on the technique's complexity [40]. This complexity implies that techniques with high learnability are more favorable from a cost perspective since they require less training.

Furthermore, adoption of test automation is associated with organizational changes, e.g. new or changed roles, which adds additional costs, especially if

the organizational changes affect the company's processes, e.g. due to changes of the intended users' responsibilities. Additionally, many automated test techniques have prerequisites that prohibit their use to certain systems written in specific programming languages, operating systems and platforms [4,11–13]. Therefore it is necessary to perform a pilot project to (1) evaluate if the new technique is at all applicable for the intended SUT and (2) for what types of tests the technique can be used. Thus a pilot project is an important activity but also associated with a, sometimes substantial, cost. However, several of these costs are often overlooked in practice and are thereby "hidden" costs associated with any change to a software process.

***Second***, for established systems, and particularly legacy systems, a considerable cost of adopting a new test technique is associated with the development of a suitably large test suite that provides test coverage of the SUT. Hence, since automated testing is primarily used for regression testing, test coverage, as stated in Section 1.2.1, is required for the testing to be efficient and valuable in finding defects.

However, this brings us to the ***third*** cost associated with automated testing which is maintenance of test scripts. Maintenance constitutes a continuous cost for all automated testing that grows with the size of the test suite. This maintenance is required to keep the test scripts aligned with the SUT's requirements [49], or at least its behavior, to ensure that test failures are caused by defects in the SUT rather than intended changes to the SUT itself, i.e. failures referred to as false positives. However, larger changes to the SUT can occur and the resulting maintenance costs can, in a worst case, become unreasonable [12]. These costs can however be mitigated through engineering best practices, e.g. modular test design [16, 40, 50]. However, best practices takes time to acquire, for any technique, and are therefore often missing, also for VGT.

Hence, these three costs must be compared together to the value provided by the automated tests, for instance value in terms of defects found or to the costs compared to alternative test techniques, e.g. manual testing. The reason for the comparison is to identify the point in time when the costs of automation break even with the alternatives, i.e. when return on investment (ROI) is achieved. Hence, for any automated test technique to be feasible, the adoption, development and maintenance costs must provide ROI and it should do so as quickly as possible. Consequently, an overall view of costs, value and other factors, e.g. learnability, adoptability and usability, is required to provide an answer if a test automation technique is applicable and feasible in practice. These factors were therefore evaluated during the thesis work to provide industrial practitioners with decision support of when, how and why to adopt and use VGT.

In summary, automated testing helps improve SUT quality and lower project costs [40]. However, the costs of automated testing can still be substantial and must therefore be evaluated against other alternative techniques to identify when and if the adoption of a new technique provides positive ROI.

### 1.2.3 Automated GUI-based Software Testing

Automated software testing has several benefits over manual testing, e.g. improved test frequency, but there are also challenges, for instance, that most techniques operate on a lower level of system abstraction. However, there is a set of automated test techniques that operate against, or through, the SUT's GUI that can be used for higher level testing. To clarify the differences between these types of GUI-based testing techniques we have divided them into three chronologically defined generations [51]. The difference between each generation is how they interact with the SUT, i.e. with exact coordinates, through hooks into the SUT's GUI or image recognition. The following section presents key properties of the three generations to provide the reader with contextual information for the continuation of the thesis.

#### 1.2.3.1 $1^{st}$ generation: Coordinate-based

$1^{st}$ generation GUI-based test automation uses exact coordinates on the screen to interact with the SUT [3]. These coordinates are acquired by recording manual interaction with the SUT and are then saved to scripts that can be replayed for automated regression testing, which improves test frequency. However, the technique is fragile, even minor changes to a GUI's layout can cause an entire test suite to fail, resulting in frequent and costly maintenance [3,52,53]. Therefore, the technique has mostly been abandoned in practice but is commonly integrated as one basic component into other test automation frameworks and tools, e.g. JUnit [53] and Sikuli [54]. However, because of the technique's limited stand-alone use in practice it will not be discussed to any extent in this thesis.

#### 1.2.3.2 $2^{nd}$ generation: Component/Widget-based

$2^{nd}$ generation GUI-based testing tools stimulate and assert the SUT through direct access to the SUT's GUI components or widgets by hooks into the SUT, e.g. into its GUI libraries or toolkits [12]. Synonyms for this technique are Component-, Widget- or Tag-based GUI testing and is performed in industrial practice with tools such as Selenium [55], QTP [56], etc.

These tools can achieve robust test case execution, e.g. few false test results, due to the tools' access and tight coupling to the SUT's internal workings, e.g. GUI events and components' ID numbers, labels, etc. These GUI events can also be monitored in a few tools to automatically synchronize the test script with the SUT, which would otherwise require the user to manually specify synchronization points in the scripts, e.g. static delays or delays based on GUI state transitions. Synchronization is a common challenge for all GUI-based test techniques because the test scripts run asynchronously to the SUT.

Another advantage of SUT access is that some of these tools can improve test script execution time by forcing GUI state transitions and bypass cosmetic, timed, events such as load screens, etc.

Further, most $2^{nd}$ generation tools support record and replay, which lowers test development costs. In addition, most tools support the user by managing GUI components' property data, e.g. ID numbers, labels, component types,

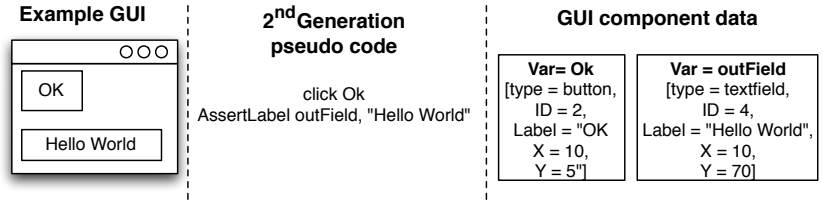


Figure 1.2: *Pseudocode example of a $2^{nd}$ generation test script for a simple application where GUI components are identified, in this case, through their properties (Tags) associated with a user defined variable.*

etc [57]. This functionality is required since these properties are unintuitive without technical or domain knowledge, e.g. an ID number or component type is not enough for a human to intuitively identify a component. However, combined, groups of properties allow the tester to distinguish between components, exemplified with pseudocode in Figure 1.2.

Some $2^{nd}$ generation tools, e.g. GUITAR [58], also support GUI ripping that allow the tools to automatically extract GUI components, and their properties, from the SUT's GUI and create a model over possible interactions with the SUT. These models can then be traversed to generate scenarios of interactions that can be replayed as test cases, a technique typically referred to as model-based testing [59–63]. As such, provided that the interaction model contains all GUI components, it becomes theoretically possible to automatically achieve full feature coverage of the SUT since all possible scenarios of interactions can be generated. However, in practice this is not possible since the number of test cases grow exponentially with the number of GUI components and length of test cases that makes it unreasonable to execute all of them. This problem is referred to as the state-space explosion problem and is common to most model-based testing tools [59]. One way to mitigate the problem is to limit the number of interactions per generated test scenario but this practice also limits the tests' representativeness of real world use and stifles their ability to reach faulty SUT states.

Furthermore, because $2^{nd}$ generation GUI-based tools' interact with the SUT through hooks into the GUI, these tests do not verify that the pictorial GUI conforms to the SUT's requirements, i.e. neither that its appearance is correct or that human interactions with it is possible. In addition, the tools require these hooks into the SUT to operate, which restricts their use to SUT's written in specific programming languages and for certain GUI libraries/toolkits. This requirement also limits the tools' use for testing of systems distributed over several physical computers, cloud based applications, etc., where the SUT's hooks are not accessible.

Another challenge is that the tools need to know what properties a GUI component has to stimulate and assert its behavior. Standard components, included in commonly used GUI libraries, e.g. JAVA Swing or AWT, are generally supported by most tools. However, for custom built components, e.g. user defined buttons, the user has to create custom interpreters or hooks for the tools to operate. However, these interpreters need to be maintained if the components are changed, which adds to the overall maintenance costs. Overall

maintenance costs that have been reported to, in some cases, be substantial in practice [10, 12, 16, 52].

However, there are also some types of GUI components that are difficult or can not be tested with this technique, e.g. components generated at runtime, since their properties are not known prior to execution of the system. As such, there are several challenges associated with $2^{nd}$ generation GUI-based testing that limit the technique's flexibility of use in industrial practice.

In summary, $2^{nd}$ generation GUI-based testing is associated with quick and often robust test execution due to their access to the SUT's inner workings. However, this access is a prerequisite for the technique's use that also limits its tools to test applications written is certain programming languages, with certain types of components, etc. As a consequence, the technique lacks flexibility in industrial use. Further, the technique does not operate on the same level of system abstraction as a human user and does therefore not verify that the SUT is correct from a pictorial GUI point of view, neither in terms of appearance or behavior. Additionally, the technique is associated with script maintenance costs that can be extensive and in worst cases infeasible [10, 12, 16, 52]. Consequently, $2^{nd}$ generation GUI-based testing does not fully fulfill the industry's needs for a flexible and feasible test automation technique.

#### 1.2.3.3 $3^{rd}$ generation: Visual GUI Testing

$3^{rd}$ generation GUI-based testing is also referred to as Visual GUI Testing (VGT) [64], and is defined as *a tool driven automated test technique where image recognition is used to interact with, and assert, a system's behavior through its pictorial GUI as it is shown to the user in user emulated system or acceptance tests.* The foundation for VGT was established in the early 90s by a tool called Triggers [18], later in the 90s accompanied by a tool called VisMap [19], which both supported image recognition based automation. However, at the time, lacking hardware support for the performance heavy image recognition algorithms made these tools unusable in practice [65]. Advances in hardware and image recognition algorithm technology have now mitigated this challenge [66] but it is still unknown if VGT, as a technique, is mature enough for industrial use. Thus providing one motivation the work presented in this thesis.

Several VGT tools are available in practice, both open source; Sikuli [20], and commercial; JAutomate [67], EggPlant [68] and Unified Functional Testing (UFT) [56], each with different benefits and drawbacks due to the tools' individual features [67]. However, common to all tools is that they use image recognition to drive scripts that allow them to be used on almost any GUI-driven application, regardless of implementation, operating system or even platform. As a consequence, VGT is associated with a high degree of flexibility. The technique does however only have limited usefulness for non-GUI systems, e.g. server-applications.

VGT scripts are written, or recorded, as scenarios that contain methods which are usually synonyms for human interactions with the SUT, e.g. mouse and keyboard events, and bitmap images. These images are used by the tools' image recognition algorithms to stimulate and assert the behavior of SUT through its pictorial GUI, i.e. in the same way as a human user. Consequently,

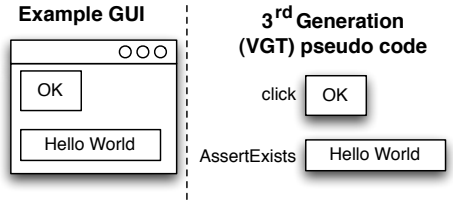


Figure 1.3: *Pseudocode example of a 3rd generation (VGT) test case for a simple application. GUI components are associated with the application's GUI component images (Bitmaps).*

VGT scripts are generally intuitive to understand, also for non-technical stakeholders, since the scripts' syntax is relatable to how the stakeholders would themselves interact with the SUT [20], e.g. click on a target represented by a bitmap and type a text represented by a string. This intuitiveness also provides VGT with high learnability also by technically awkward users [65].

A pseudo-code VGT script example is shown in Figure 1.3 that performs the same interactions as the example presented for $2^{nd}$ generation GUI-based testing, presented in Figure 1.2, for comparison.

Conceptually, image recognition is performed in two steps during VGT script playback. First, the SUT's current GUI state is captured as a bitmap, e.g. in a screenshot of the computers desktop, which is sent together with the sought bitmap from the VGT script to the image recognition algorithm. Second, the image recognition algorithm searches for the sought bitmap in the screenshot and if it finds a match it returns the coordinates for the match that are then used to perform an interaction with the SUT's GUI. Alternatively, if the image recognition fails, a false boolean is returned or an exception is raised.

Different VGT tools use different algorithms but most algorithms rely on similarity-based matching which means that a match, i.e. sought bitmap, is found if it is within a percentile margin between the identified and sought bitmap image [20]. This margin is typically set to 70 to 80 percent of the original image to counteract failures due to small changes to a GUI's appearance, e.g. change of a GUI bitmap's color tint. However, similarity-based matching does not prevent image recognition failure when bitmaps are resized or changed completely.

Additionally, VGT scripts, similar to $1^{st}$ and $2^{nd}$ generation scripts, need to be synchronized with the SUT's execution. Synchronization in VGT is performed with built in functionality or methods that wait for a bitmap(s) to appear on the screen before the script can proceed. However, these methods also make VGT scripts slow since they cannot execute quicker than the state transitions of the GUI, which is a particular challenge for web-systems since waits also need to take network latency into account.

In summary, VGT is a flexible automated GUI-based test technique that uses tools with image recognition to interact and assert a SUT's behavior through its pictorial GUI. However, the technique's maturity is unknown and this thesis therefore aims to evaluate if VGT is applicable and feasible in industrial practice.

#### 1.2.3.4 Comparison

To provide a general background and overview of the three generations of automated GUI-based testing, some of their key properties have been presented in Table 1.1. The table shows which properties that each technique has ("Y") or not ("N") or if a property is support by some, but not all, of the technique's tools ("S"). These properties were acquired during the thesis work as empirical results or through analysis of related work. However, they are not considered to be part of the thesis main contributions even though they support said contributions.

Several properties are shared by all techniques. For instance, they can all be used to automate manual test cases but only VGT tools also support bitmap assertions and user emulation and it is therefore the only technique that provides results of equal quality to manual tests. Further, all three techniques are perceived to support daily continuous integration and all techniques require the scripts to be synchronized with the SUT's execution. Finally, none of the techniques are perceived as replacements to manual testing since all of the techniques are designed for regression testing and therefore only find defects in system states that are explicitly asserted. In contrast, a human can use cognitive reasoning to determine if new, previously unexplored, states of the SUT are correct. Consequently, a human oracle [69] is required to judge if a script's outcome is correct or not.

Other properties of interest regard the technique's robustness to change. For instance, both $2^{nd}$ and $3^{rd}$ generation tools are robust to GUI layout change, assuming, for the $3^{rd}$ generation, that the components are still shown on the screen after change. In contrast, $1^{st}$ generation tools are fragile to this type of change since they are dependent on the GUI components' location being constant.

However, $1^{st}$ generation tools, and also $3^{rd}$ generation tools, are robust to changes to the SUT's GUI code whilst $2^{nd}$ generation tools are not, especially if these changes are made to custom GUI components, the GUI libraries or GUI toolkits [12].

Finally, $1^{st}$ and $2^{nd}$ generation tools are robust to changes to the GUI components' bitmaps since none of the techniques care about the GUI's appearance. In contrast, $3^{rd}$ generation tools fail if either the appearance or the behavior of the SUT is incorrect.

Consequently, the different techniques have different benefits and drawbacks that are perceived to make the techniques more or less applicable in different contexts.

## 1.3 Research problem and methodology

In this section, a summary of the background and the motivation for the research performed in this thesis work are presented. These are based on the challenges and gaps in knowledge and tooling presented in Sections 1.1 to 1.2.3.4. Additionally, the research objective is presented and broken down into four specific research questions that the thesis work aimed to answer through an incremental research process that is also presented. Finally, the research methodology and research methods used during the thesis work are discussed.

| **Property** | **1$^{st}$ Gen.** | **2$^{nd}$ Gen.** | **3$^{rd}$ Gen.** |
|---|---|---|---|
| Independent of SUT platform | N | N | Y |
| Independent of SUT programming language | Y | S | Y |
| Non-intrusive test execution | N | S | Y |
| Emulates human user behavior | Y | N | Y |
| Open-source tool alternatives | Y | Y | Y |
| Supports manual test case automation | Y | Y | Y |
| Supports testing of custom GUI components | Y | S | Y |
| Supports bitmap-based assertions | N | S | Y |
| Supports testing of distributed systems | Y | S | Y |
| Supports daily continuous integration | Y | Y | Y |
| Robust to GUI layout change | N | Y | Y |
| Robust to system code change | Y | N | Y |
| Robust to bitmap GUI component change | Y | Y | N |
| Support script recording (as opposed to manual scripting) | Y | Y | S |
| Script execution time independent of SUT performance | N | N | N |
| Replacement of other manual/automatic test practices | N | N | N |

Table 1.1: *The positive and negative properties of different GUI-based test techniques. All properties have been formulated such that a "Y" indicates that the property is supported by the technique. "N" indicates that the property is not supported by the technique. "S" indicates that some of the technique's tools supports the property, but most don't.*

### 1.3.1 Problem background and motivation for research

***Background:*** Testing is the primary means by which companies verify and validate (V&V) their software. However, the costs of V&V ranges between 20-50 percent of the total costs associated with a software development project [1, 28,29], which is a challenge that can be contributed to the extensive industrial use of manual, tedious, time consuming, and therefore error prone V&V practices [2–7]. Automated testing is generally proposed as the solution to this challenge, since automated test scripts execute systematically each time and with reduced human effort and cost [40]. However, this proposition presents new challenges for software development companies, such as *what automated testing do they need, how is it performed and how does it provide value?*

The most common type of automated testing in practice is automated unit testing [14, 33], which has been shown to be effective to find software defects. However, unit tests operate on a low level of system abstraction and they have therefore been debated to be ill suited for V&V of high level requirements [15,16]. Automated unit testing therefore has a place in software development practice but should be complemented with test techniques also on higher levels of system abstraction to provide full automated coverage of the SUT [16]. For GUI-driven software this also includes automated testing of the pictorial GUI as shown to the user.

To acquire GUI automation coverage, many companies use $2^{nd}$ generation GUI-based testing for automated system testing, for instance with the tool Selenium [55]. However, these tools interact with the SUT by hooking into its GUI libraries, toolkits or similar and therefore do not verify that human interaction with the SUT's pictorial GUI can be performed as expected [51]. Such verification requires an automated test technique that can operate on the same level of abstraction and with the same confidence and results as a human user.

In addition, most automated test techniques' are restricted to be used on SUTs that fulfill the tools' prerequisites, such as use of specific programming languages, platforms, interfaces for testing etc [4, 11–13]. These prerequisites are a particular challenge for legacy, or distributed, systems that are either not designed to support automated testing or lack the necessary interfaces for test automation. As a consequence, industry is in need of a flexible test automation technique with less, or easily fulfilled, prerequisites.

Further, the view that automated testing lowers test related cost is only partially true because test automation is still associated "hidden" costs and, in particular, maintenance costs [10, 12, 16, 40, 52]. Therefore, adoption of automated testing can lower the total development cost of a project by enabling faster feedback to developers that leads to faster defect resolution, but test related costs still remain or can even increase. As such, to fulfill industry's need for a flexible GUI-based test automation technique, a technique must be identified that is feasible long-term and which preferably provides quick ROI compared to manual testing. Such a technique must also provide value in terms of, at least, equal defect finding ability as manual testing and with low test execution time to facilitate frequent test execution.

***Motivation:*** In theory, Visual GUI Testing (VGT) fulfills the industrial need for a flexible, GUI-based, automated test technique due to its unprece-

| Paper | Objective | RQ1 | RQ2 | RQ3 | RQ4 |
|---|---|---|---|---|---|
| A | Static evaluation of VGT in practice | X | X | X | |
| B | Dynamic evaluation of VGT in practice | X | X | X | |
| C | Challenges, problems and limitations with VGT in practice | X | X | X | |
| D | Maintenance and return on investment of VGT | X | X | | |
| E | Long-term use of VGT in practice | X | X | X | |
| F | Model-based VGT combined with $2^{nd}$ generation GUI-based testing | X | | | X |
| G | Failure replication | X | | | X |

Table 1.2: *Mapping of research questions to the individual publications presented in this thesis.*

dented ability to emulate human interaction and assertions through a SUT's pictorial GUI, an ability provided by the technique's use of tools with image recognition. However, the technique's body of knowledge is limited, in particular in regards to empirical evidence for its applicability and feasibility of use in industrial practice. This lack of knowledge is the main motivator for the research presented in this thesis since such knowledge is required as decision support for industrial practitioners to evaluate if they should adopt and use the technique. Consequently, this research is motivated by an industrial need for a flexible and cost-effective GUI-based test automation technique that can emulate end user behavior with at least equal defect-finding ability as manual testing but with lower test execution time. From an academic point of view, the research is also motivated since it provides additional empirical evidence from industry regarding the adoption, use and challenges related to automated testing.

***Research Objective:*** The objective of this thesis is to identify empirical evidence for, or against, the applicability and feasibility of VGT in industrial practice. Additionally, to identify what challenges, problems and limitations that impede the technique's short and long-term use. Hence, an overall view of the current state-of-practice of VGT, including alternative and future application areas for the technique. Consequently, knowledge that can be used for decision support by practitioners and input for future academic research.

***Research questions:*** The research objective was broken down into four research questions presented below together with brief descriptions of how they were answered. Further, Table 1.2 presents a mapping between each research question and the papers included, and presented later, in this thesis.

**RQ1:** What key types of contexts and types of testing is Visual GUI Testing generally applicable for in industrial practice?

This question addresses the rudimentary capabilities of VGT, i.e. can the tech-

nique at all find failures and defects on industrial grade systems? Additionally, it aims to identify support for what types of testing VGT is used for, e.g. only regression testing of system and acceptance tests or exploratory testing as well? This question also addresses if VGT can be used in different contexts and domains, such as agile software development companies, for safety-critical software, etc. Support for this question was acquired throughout the thesis work but in particular in the studies presented in Chapters 2, 3, 4, 6 and 8, i.e. *Papers A, B, C, E and G.*

**RQ2:** To what extent is Visual GUI Testing feasible for long-term use in industrial practice?

Feasibility refers to the maintenance costs and return on investment (ROI) of adoption and use of the technique in practice. This makes this question key to determine the value and long-term industrial usability of VGT. Hence, if maintenance is too expensive, the time to positive ROI may outweigh the technique's benefits compared to other test techniques and render the technique undesirable or even impractical in practice. This question also concerns the execution time of VGT scripts to determine in what contexts the technique can feasibly be applied, e.g. for continuous integration? Support for this research question was, in particular, acquired in three case studies at four different companies, presented in Chapters 3, 5, and 6, i.e. *Papers B, D and E.*

**RQ3:** What are the challenges, problems and limitations of adopting, using and maintaining Visual GUI Testing in industrial practice?

This question addresses if there are challenges, problems and limitations (CPLs) associated with VGT, the severity of these CPLs and if any of them prohibit the technique's adoption or use in practice. Furthermore, these CPLs represent pitfalls that practitioners must avoid and therefore take into consideration to make an informed decision about the benefits and drawbacks of the technique, i.e. how the CPLs might affect the applicability and feasibility of the technique in the practitioner's context. To guide practitioners, this question also includes finding guidelines for the adoption, use and long-term use of VGT in practice.

Results to answer this question were acquired primarily from three case studies that, fully or in part, focused on CPLs associated with VGT, presented in Chapters 3, 4 and 6, i.e. *Papers B, C and E.*

**RQ4:** What technical, process, or other solutions exist to advance Visual GUI Testing's applicability and feasibility in industrial practice?

This question refers to technical or process oriented solutions that improve the usefulness of VGT in practice. Additionally, this question aims to identify future research directions to improve, or build upon, the work presented in this thesis.

Explicit work to answer the question was performed in an academic study, presented in Chapter 7, i.e. *Paper F*, where VGT was combined with $2^{nd}$ generation technology to create a fully automated VGT tool. Additional support was acquired from an experience report presented in Chapter 8 (*Paper G*) where a novel VGT-based process was reported from industrial practice.

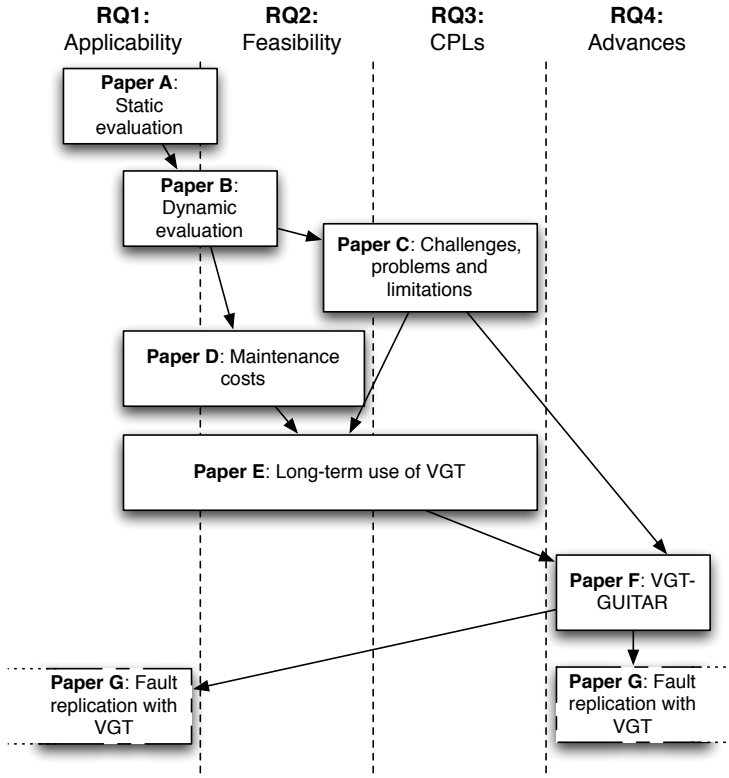


Figure 1.4: *A chronological mapping of how the studies included in this thesis are connected to provide support for the thesis four research questions. The figure also shows which papers that provided input (data, challenges, research questions, etc.) to proceeding papers.* ***CPLs*** *- challenges, problems and limitations.*

### 1.3.2 Thesis research process

Figure 1.4 presents an overview of the incremental research process that was used during the thesis work and how included research papers are connected. These connections consist of research results or new research questions that were acquired in a study that required, or warranted, additional research in later studies.

The thesis work began with a static evaluation of VGT (*Paper A*) that provided initial support for the applicability and costs associated with VGT. Next, VGT was evaluated dynamically in an industrial project (*Paper B*) where VGT was adopted and used by practitioners. This study provided additional information about the applicability and initial results about the feasibility of VGT. In addition, challenges, problems and limitations (CPLs) were identified that warranted future research that was performed in *Paper C*. *Paper C* concluded that there are many CPLs associated with VGT but none that prohibit its industrial use. Therefore, the thesis work proceeded with an evaluation of the feasibility of VGT in an embedded study where results regarding the long-term maintenance costs and return on investment (ROI) of VGT were acquired (*Pa-*

*per D*). These results were acquired through empirical work with an industrial system (Static analysis) and interviews with practitioners that had used VGT for several months (Dynamic analysis). However, results regarding the long-term feasibility of the technique were still missing, a gap in knowledge that was filled by an interview study at a company that had used VGT for several years (*Paper E*). Consequently, these studies provided an overall view of the current state-of-practice of VGT. In addition they provided support to draw conclusions regarding the applicability (***RQ1***) and feasibility (***RQ2***) of VGT in practice but also what CPLs that are associated with the technique (***RQ3***).

Further, to advance state-of-practice, a study was performed where VGT was combined with $2^{nd}$ generation technology that resulted in a building block for future research into fully automated VGT (*Paper F*)(***RQ4***). Additional support for ***RQ4*** was acquired from an experience report from industry (*Paper G*) where a novel semi-automated exploratory test process based on VGT was reported.

Combined, these studies provide results to answer the thesis four research questions and a significant contribution to the body of knowledge of VGT and automated testing.

### 1.3.3 Research methodology

A research methodology is a structured process that serves to acquire data to fulfill a study's research objectives [70]. On a high level of abstraction, a research process can be divided into three phases: preparation, collection and analysis (PCA). In the preparation phase the study's research objectives, research questions and hypotheses are defined, including research materials, sampling of subjects, research methods are chosen for data collection, etc. Next, data collection is performed that shall preferably be conducted with several methods and/or sources of evidence to enable triangulation of the study's results and improve the research validity, i.e. the level of trust in the research results and conclusions [70–72]. Finally, in the analysis phase, the acquired research results are scrutinized, synthesized and/or equated to draw the study's conclusions that can be both positive or negative answers to a study's research question(s).

Some research methodologies deviate from the PCA pattern and are instead said to have a flexible design. Flexible design implies that changes can be made to the design during the study to, for instance, accommodate additional, unplanned, data collection opportunities [17].

A researcher can create an ad hoc research methodology if required, but several common methodologies exist that are used in software engineering research, e.g. case studies [17], experiments [73] and action research [74].

Two research methodologies were used extensively during this thesis work: case studies and experiments. This choice was motivated by the thesis research questions and the studies' available resources. Action research was, for instance, not used because it requires a longitudinal study of incremental change to the studied phenomenon which makes it resource intensive and places a larger requirement on the collaborating company's commitment to the study. Hence, a commitment that many companies are reluctant to give to an immature research area such as VGT.

Research methodologies have different characteristics and thus, inherently, provide different levels of research validity [72]. Validity is categorized in different ways in different research fields but in this thesis it is categorized according to the guidelines by Runeson and Höst [17], into the following categories:

- **Construct validity** - The suitability of the studied context to provide valid answers to the study's research questions,
- **Internal validity** - The strength of cohesion and consistency of collected results.
- **External validity** - The ability to generalize the study's results to other contexts and domains, and
- **Reliability/Conclusion validity** - The degree of replicability of the study's results.

Case studies provide a deeper understanding of a phenomenon in its actual context [17] and therefore have inherently high construct validity. In addition, given that a case study is performed in a well chosen context with an appropriate sample of subjects, it also provides results of high external validity. However, case studies in software engineering are often performed in industry and are therefore governed by the resources provided by the case company, which limits researcher control and can negatively affect the results internal validity.

In contrast, experiments [73] are associated with a high degree of researcher control. This control is used to manipulate the studied phenomenon and randomize the experimental sample to mitigate factors that could adversely affect the study's results. As such, experiments have inherent high internal validity but it comes at the expense of construct validity since the studied phenomenon is, by definition, no longer studied in its actual context. In addition, similar to case studies, the external validity of experimental results depend on the research sample.

Furthermore, research methodologies can be classified based on if they are qualitative or quantitative [70], where case studies are associated with qualitative data [17], e.g. data from interviews, observations, etc., and experiments are associated with quantitative data [73], e.g. measurements, calculations, etc. These associations are however only a rule of thumb since many case studies include quantitative data to support the study's conclusions [73] and experiments often support their conclusions with qualitative observations. During the thesis work, both types of data were extensively used to strengthen the papers', and the thesis, conclusions and contributions. This strength is provided by quantitative results' ability to be compared between studies, whilst qualitative data provides a deeper understanding of the results.

### 1.3.4 Case studies

A case study is defined as a study of a phenomenon in its contemporary context [17, 71]. The phenomenon in its context is also referred to as the study's unit of analysis, which can be a practice, a process, a tool, etc., used in an

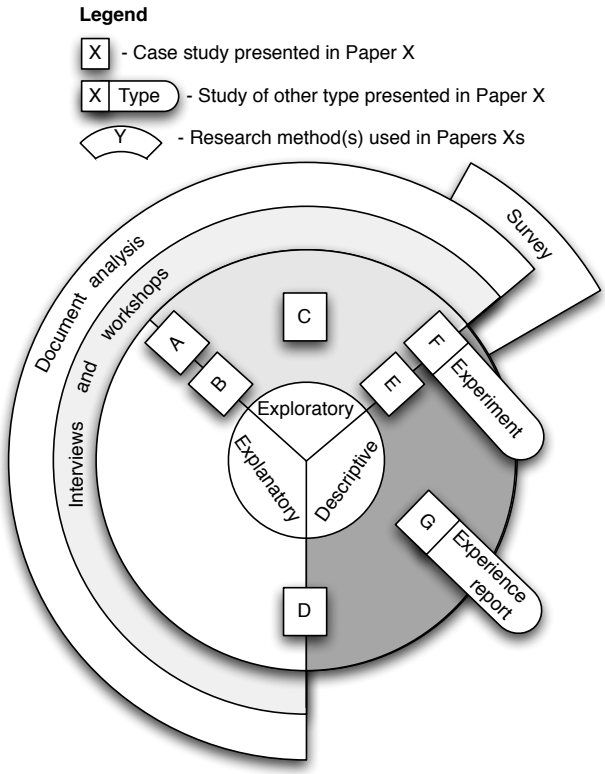


Figure 1.5: *Visualization of the categorization of each of the included papers.*

organization, company or similar context. Case studies are thereby a versatile tool in software engineering research since they can be tailored to certain contexts or research questions and also support flexible design [17]. Additionally, case studies can be performed with many different research methods, e.g. interviews, observations, surveys, etc [17].

Further, case studies can be classified as *single* or *multiple* and *holistic* or *embedded* case studies, where single/embedded refer to the number of contexts in which the unit (holistic) or units (embedded) of analysis are studied [71].

Case study results are often anecdotal evidence, e.g. interviewees' perceptions of the research phenomenon, which makes triangulation an essential practice to ensure result validity [17, 71]. Further, case studies should be replicable, which implies that all data collection and analysis procedures must be systematic and thoroughly documented, for instance in the form of a case study protocol [71], to establish a clear chain of evidence. A more detailed discussion about analysis of qualitative data is presented in Section 1.3.6.

Case studies were the primary means of data collection for this thesis and were conducted with, or at, software development companies in Sweden. These companies include several companies in the Saab corporation, Siemens Medical and Spotify. The first case studies, reported in *Papers A, B, C*, were exploratory, continued with *Paper D* that was explanatory and concluded with

*Paper E* that was descriptive, depicted in Figure 1.5. Hence, the thesis work transitioned from exploration to explanation of the capabilities and properties of VGT to description of its use in practice. This transition was driven by the incrementally acquired results from each study, where later studies thereby aimed to verify the results of earlier studies. Figure 1.5 also includes studies that were not case studies, i.e. *Papers F and G* which were an experiment and an experience report respectively, depicted to show how they were classified in relation to the other papers included in the thesis.

Furthermore, the performed case studies were all inherently different, i.e. conducted with different companies, in different domains, with different subjects and VGT tools, which has strengthened both the construct and external validity of the thesis conclusions. Further, interviews were used for the majority of the data collection to acquire in depth knowledge about the adoption and use of VGT. However, quantitative, or quantifiable, data was also acquired since it was required to compare VGT to other test techniques in the studies, and the thesis. For instance, quantitative data was acquired to compare the performance and cost of VGT to both manual test techniques and $2^{nd}$ generation GUI-based testing. However, comparisons were also made with qualitative data, such as practitioners' perceptions about benefits and drawbacks of different techniques, to get a broad view of the techniques' commonalities and differences in different contexts. Thus ensuring that the included studies' individual contributions were triangulated with data from different sources and methods to improve the results internal validity.

#### 1.3.4.1 Interviews

Interviews are commonly used for data collection in case study research and can be divided into three different types: structured-, semi-structured and unstructured interviews [71, 75]. Each type is performed with an interview guide that contains step-by-step instructions for an interview, including the interview questions, research objectives, etc. In addition, interview guides shall include a statement regarding the purpose of the study and insurance of the interviewee's anonymity, which helps to mitigate biased or untruthful answers. Further, these types of interviews vary in strictness, which relates to the makeup of the interview guide as well as the interviewer's freedoms during an interview.

***Structured interviews:*** Structured interviews are the most strict [71] and restrict the interviewer from asking follow up questions or ask the interviewee to clarify their answers. Therefore, considerable effort should be spent on the interview guide to test it and to ensure that the interview questions are unambiguous and valid to answer the study's research questions. Structured interview questions can be of different type but multiple-choice or forced-choice are the most common. Forced choice questions, e.g. Likert-scale questions, can be analyzed with statistics [76] but require a larger interview sample, which makes the method costly in terms of resources. Therefore, structured interviews were not used during the thesis work.

***Semi-structured interviews:*** The second most strict type of interview is called semi-structured interviews [71], which allow the interviewer to elicit more in depth or better quality information by asking follow up questions or

by clarifying questions to the interviewee. These interviews are therefore suitable in descriptive studies, where the studied phenomenon is partly known, or exploratory studies, where little or nothing is known about the phenomenon. In both cases, several interviews should be performed where each interview should add to the researcher's understanding of the studied phenomenon and allow the researcher to tailor each consecutive interview, i.e. change the interview questions, to acquire more in depth knowledge. However, care should be taken when changes are made to ensure that the interview results can still be triangulated, i.e. the interview questions must not diverge too much between interviews.

Semi-structured interviews were extensively used during the thesis work both to explore and describe the use of VGT and its associated CPLs [17]. Further, interview guides were always used but they were seldom changed between interviews, instead more in depth information was acquired through ad hoc follow-up questions in each interview. The baseline of common questions was kept to make triangulation of interview results easier. These interviews were all recorded, transcribed and analyzed with less rigorous qualitative analysis or Grounded Theory analysis [77], i.e. rigorous coding analysis [78].

***Unstructured interviews:*** Finally, the least strict type of interviews are unstructured interviews [71] where interview guides are optional, but recommended, to guide the interview. This type of interview is therefore suitable for exploratory studies [17] and were primarily used in the thesis work in pre-studies to acquire contextual information about the research companies through general questions regarding the companies processes, practices, organizations, etc. In addition to context information, they also provided baseline information for additional interviews.

#### 1.3.4.2 Workshops

Workshops are often performed with groups of participants, similar to focus groups [79] and their purpose is to explore, describe or explain a phenomenon under study [80] through discussions, brainstorming [81], activity participation, etc. As such, workshops can be a means to acquire both in depth and broad information in a short amounts of time, as discussed by Young for the purpose of requirements elicitation [82].

However, several prerequisites must be fulfilled for a workshop to be successful, for instance, the workshop participants must be a well chosen sample, i.e. with participants of varying experience, roles, age, gender, etc. This prerequisite can be challenging to fulfill in industrial studies since access to suitable participants is often restricted, for instance due to resource or scheduling constraints, etc. In addition, some constellations of participants can add bias to the results. For instance, employees can be reluctant to give their honest opinions if their managers are present in the workshop which presents a threat to the results. Additionally, if a sample is too uniform it may not provide representative results for the whole company or other domains, which is a threat to the results external validity. Another challenge with workshops is to keep them in scope of the study, especially if lively, and perhaps interesting, discussions occur, and they should therefore be moderated. Because of these challenges, workshop results must also be triangulated with, for instance,

follow-up interviews, surveys, etc.

A workshop can be designed ad hoc to achieve a specific research objective, or performed with a standard workshop format, e.g. KJ-Shiba [83]. Regardless, workshops must be thoroughly planned. This planning includes sampling of suitable participants, creation of workshop materials, planning the analysis procedure, etc.

Workshops were used in several studies included in the thesis, for two reasons. First, to quickly acquire information about a research company's contexts, practices and tools. Hence, exploratory workshops with one or several participants where unstructured or semi-structured interviews and visual aids, e.g. white-board drawings, were commonly used. Second, workshops were used for result verification and triangulation where workshops began with a presentation of the study's acquired results followed by discussion and analysis of key results with the workshop's participants. These workshops, presented in *Papers B, C and E*, were well planned, with predefined workshop materials such as interview questions, presentation materials, etc., but participants were mostly sampled with convenience sampling due to resource constraints.

#### 1.3.4.3 Other

Interviews and workshops were the primary methods used in the case studies included in this thesis. However, other methods were also used during the thesis work[1], some of which will be briefly discussed in this section.

***Document analysis:*** Interviews and workshops acquire first degree data [71, 84], i.e. data directly from a source of information such as an interviewee. In turn, second degree data is collected indirectly from a source, for instance, through transcription of a recorded interview. However, document analysis relies on third degree data [17], which is data that has already been transcribed and potentially analyzed.

From a company perspective, document analysis can be a cost-effective method of data transference but can be a time-consuming activity for the researcher, especially in foreign or highly technical domains. Further, third degree data is created by a third person and can therefore include biases which means that document root-source analysis is required to identify who created the document, for what purpose, the age of the information, etc. [17], i.e. to evaluate the documented information's validity.

Document analysis was used in the thesis work to acquire information about the research companies' processes and practices. In particular, test specifications were analyzed to give input for empirical work with VGT at the studied companies, e.g. in *Paper A* where manual test cases at Saab AB were automated with two different VGT tools.

Further, this method can be used in a survey to conduct systematic mappings and systematic literature reviews [85] of published research papers. However, due to the limited body of knowledge on VGT, no such study was performed during the thesis work.

***Surveys:*** Surveys are performed on samples of people, documents, software, or other group [86] for the purpose of acquiring general conclusions

[1] In this instance, thesis work refers also to studies performed by the author but not included in the thesis.

regarding an aspect of the sample [71]. For instance, a survey with people can serve to acquire their perceptions of a phenomenon, document surveys instead aim at document synthesis [85], etc.

In software engineering research, surveys are often performed with questionnaires as an alternative to structured interviews [71]. One benefit of questionnaires is that they can be distributed to a large sample at low cost but if there is no incentive for the sample to answer the questionnaire, the participant response-rate can be low, i.e. less than the rule of thumb of 60 percent that is suggested for the survey to be considered valid.

Questionnaire questions can be multiple-choice, forced-choice or open, i.e. free text. Forced choice questions are often written as Likert scale questions [76], i.e. on an approximated ratio-scale between, for instance, totally disagree and totally agree. In turn, multiple choice questions can ask participants to rank concepts on ratio-scales, e.g. with the 100 dollar bill approach [87]. However, questions can have other scales such as nominal, ordinal or interval scales [88]. These scales serve different purposes and it is therefore important to choose the right type to be able to answer the study's research questions. Regardless, questionnaire creation is a challenge since the questions must be unambiguous, complete, use context specific nomenclature, etc., to be of high quality. Therefore, like interview guides, questionnaires must be reviewed and tested prior to use.

Questionnaires were used during the thesis work to verify previously gathered results and to acquire data in association with workshops. Explicitly, among the research papers included in this thesis, a questionnaire survey was used in *Paper E*. The results of the surveys were then analyzed qualitatively or with formal or descriptive statistics, discussed further in Section 1.3.6, to test the studies' hypotheses or answer the studies' research questions.

***Observation:*** Observations are fundamental in research and can be used in different settings and performed in different ways, e.g. structured or unstructured [89]. One way to perform an observation is the fly on the wall technique, where the researcher is not allowed to influence the person, process, etc., being observed. Another type is the talk-aloud protocol, where the observed person is asked to continuously describe what (s)he is doing [17]. As such, observations are a suitable practice to acquire information about a phenomenon in its actual context and can also provide the researcher with deeper understanding of domain-specific or technical aspects of the phenomenon.

However, observation studies are associated with several threats, for instance the Hawthorne effect, which causes the observed person to change his/her behavior because they know they are being observed [90]. Therefore, the context of the observation must be taken into consideration as well as ethical considerations, e.g. how, what and why something or someone is being observed [17]. An example of unethical observation would be to observe a person without their knowledge.

Planned observation, with an observation guide, etc., was only used once during the thesis work to observe how manual testing was performed at a company. However, observations were also used to help explain results from the empirical studies with VGT, i.e. in *Papers A and D*.

### 1.3.5 Experiments

Experimentation is a research methodology [73] that focuses on answering what factor(s) (or *independent variable(s)*) that affect a measured factor of the phenomena (*the dependent variable(s)*). As such, experiments aim to compare the impact of treatments (change of the independent variable(s)) on the dependent variable(s).

Experimental design begins with formulation of a research objective that is broken down into research questions and hypotheses. A hypothesis is a statement that can be either true or false that the study will test, for instance the expected outcome of a treatment on the dependent variable. Therefore, experiments primarily aim to acquire quantitative or quantifiable data, which can be analyzed statistically to accept or reject the study's hypotheses and answer the study's research questions.

However, experiments are also affected by confounding factors [73], i.e. factors outside the researcher's control that also influence the dependent variable. These factors can be mitigated through random sampling that cancels out the confounding factors across the sample [91] such that measured changes to the dependent variable are caused only by changes to the independent variable(s).

However, in some contexts, e.g. in industry, it is not possible to randomize the studied sample and instead quasi-experiments need to be used [73, 92]. Whilst controlled experiments are associated with a high degree of internal validity but lower construct validity (due to manipulation of contextual factors), quasi-experiments have lower internal validity but higher construct validity since they are performed in a realistic context.

Further, compared to case studies, controlled experiments have high replicability, i.e. an experiment with a valid experimental procedure can be replicated to acquire the same outcome as the original experiment. It is therefore common that publications that report experimental results present the experimental procedure in detail and make the experimental materials available to other researchers.

Experiments were performed as part of two papers included in the thesis, i.e. *Papers A and F*. In *Paper A*, the applicability of VGT to test non-animated GUIs was compared to testing of animated GUIs. Additionally in *Paper F*, to compare the false test results generated by $2^{nd}$ and $3^{rd}$ generation GUI-based testing during system and acceptance tests.

### 1.3.6 Data analysis

Research methodologies and methods provide the researcher with results that sometimes are enough to support, or answer, a study's research questions. However, in most studies the results, by themselves, are not enough and they must therefore be analyzed.

Research results can be classified as qualitative, e.g. statements, and quantitative, e.g. numbers, [70], as stated in Section 1.3.3. However, regardless of data type, results must be triangulated [17, 71], which for qualitative data can be a challenge. The reason is because qualitative methods generally produce large quantities of information that are difficult to overview and synthesize. This challenge can be solved by quantifying the information through coding [78] where statements, observations, etc., are clustered in categories

defined by codes that can then be analyzed individually to identify support for the study's research question(s). This practice is key in Grounded Theory research [77] and is recommended to acquire a strong chain of evidence [71] and can be performed in different ways. However, a general recommendation is that it is performed by several people to mitigate coding bias.

During the thesis work, coding was only used for the data analysis in *Paper F* since the study relied exclusively on qualitative data. Previous studies, included in the thesis, all include empirical work and/or quantitative data, which justified the use of less stringent analysis methods since the results of these studies were triangulated with data sources of different type.

Further, quantitative data can be analyzed statistically, either with descriptive or formal methods [93,94]. Descriptive statistics are used to visualize data to provide an overview of the data, e.g. in graphs and plots, which for larger data sets can be particularly helpful since it helps establish patterns, distributions and other insights to the data. Descriptive statistics were particularly helpful in *Paper D* to visualize and draw conclusions regarding the maintenance costs and return on investment of VGT.

In contrast, formal statistics are mathematical methods that are used to compare, correlate, or evaluate data to find statistically likely patterns, e.g. to compare if two data sets are (statistically) significantly different. Formal statistical analysis was used in several of the studies included in the thesis, i.e. *Papers A, D, E and F*, most often performed with the Student T-Test, the Wilcoxon rank-sum test or the Mann-Whitney U test [95].

Further, formal statistical methods can be split into two different categories, parametric and non-parametric tests [93, 96], where parametric tests provide statistical results of high precision but are associated with more prerequisites than non-parametric tests. These prerequisites include that the data set should be normally distributed, that the sample is of suitable size, etc. However, these prerequisites are often difficult to fulfill in industrial studies and therefore it is argued that the use of parametric tests should be avoided in favor of non-parametric tests in software engineering research [96], despite the fact that non-parametric tests lower statistical precision. Due to the industrial focus of the thesis work, most formal statistical analysis performed in the included studies were non-parametric, limited primarily by lack of normal distribution in the acquired data sets.

In summary, research results can be both qualitative and quantitative and analyzed with both descriptive and formal statistics, all used during this thesis work to strengthen the validity of the studies' conclusions. Hence, quantitative data was used to statistically compare VGT tools and VGT to other test techniques, triangulated with qualitative data to explain the quantitative results and vice versa.

| Paper | Name | Domain | Size (S/M/L) | City | Development process(es) | Test strategy | VGT tool |
|---|---|---|---|---|---|---|---|
| A, C and G | Saab AB | Safety-critical air-traffic management software | M | Gothenburg | Plan-driven and Agile | Manual system and acceptance testing | Sikuli (Python API) |
| B | Saab AB | Mission-critical military control software | M | Järfälla | Plan-driven and Agile | Manual system and acceptance testing, automated unit testing | Sikuli (Python API) |
| D | Saab AB | Safety-critical air-traffic management software | M | Växjö | Plan-driven and Agile | Manual system and acceptance testing | Sikuli (Python API) |
| D | Siemens Medical | Life-critical medical journal systems | S | Gothenburg | Agile (Scrum) | Manual scenario-based and exploratory system and acceptance testing, automated unit testing | JAutomate |
| E | Spotify | Entertainment streaming application | L | Gothenburg/ Stockholm | Agile (Scrum) | Manual scenario-based and exploratory system, acceptance and user experience testing, automated unit, integration and system testing. | Sikuli (Java API) |

Table 1.3: *Summary of key characteristics of the companies/divisions/groups that took part in the studies included in the thesis. In the column "size" the companies' contexts are categorized as small (S), medium (M) and large (L) where small is less than 50 developers, medium is less than 100 developers and large is more than 100 developers in total. Note that Saab AB refers to explicit divisions/companies within the Saab coorporation.* ***API*** *- Application Programming Interface.*

## 1.4 Overview of publications

The following section will present summaries of the studies included in this thesis, including their research objectives, methodology, results and contributions. These studies were primarily performed in Swedish industry at companies with different organizations, processes and tools that develop both safety-critical systems as well as non-safety critical applications. An overview of these companies has been presented in Table 1.3 based on a set of characteristics that was acquired for to all companies, which includes size, domain, used test techniques, etc.

### 1.4.1 Paper A: Static evaluation

Paper A, presented in Chapter 2, is titled "Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry". The paper presents a single, embedded, case study at the safety-critical air-traffic management software development company Saab AB in the Swedish city of Gothenburg.

***Research Objective:*** The main research objective of the study was to acquire initial support for the applicability of VGT in industrial practice. Specifically, its ability to automate system and acceptance tests for automated GUI-based regression testing.

***Methodology:*** The case study consisted of two phases where two VGT tools, one commercial referred to as CommercialTool[2] and an open source, called Sikuli [20, 54], were evaluated. In Phase 1, static evaluation was performed of the tools to acquire their different properties. In addition, four experiments were performed to compare the tools' image recognition algorithms' ability to test animated and non-animated GUIs.

In Phase 2, 10 percent of an industrial test suite of 50 manual test cases was automated in each tool. These test cases were carefully chosen after a manual regression test of one of Saab's main product's subsystems, in continuation called the SUT. The SUT had in the order of multiple 100K lines of code, had a non-animated GUI and a comprehensive manual test suite. During test automation, measurements were taken on the development time, lines of code and script execution time. These were then compared statistically to determine if there was any statistically significant difference between the tools.

***Results:*** Twelve (12) properties were identified in the static evaluation that showed that the two tools had different favorable properties. For instance, whilst CommercialTool had faster image recognition and support for automated test failure mitigation, Sikuli was free of charge and generally more user friendly. Further, in the experiment, CommercialTool had higher success-rate than Sikuli for non-animated GUIs but Sikuli had better success-rate for animated GUIs.

In Phase 2, the development time and execution time of the SUT's entire test suite was estimated based on the developed test scripts. These estimates showed that adoption of VGT could provide positive return on investment within one development iteration of the SUT compared to manual testing. Additionally, the estimates showed that the execution time of the suite was

[2]For reasons of confidentiality we cannot disclose the name of the tool

3.5 hours, which was an improvement of 4.5 times compared to manual testing that took 16 hours on average.

Further, none of the null hypotheses in regards to development time, lines of code and execution time could be rejected. The study therefore concludes that there is no statistical significant difference between the two tools on any of these measures. Therefore, since the tools could successfully automate the industrial test cases, the study provides initial support that VGT is applicable for automation of manual system test cases in industrial practice.

***Contributions:*** The study's main contributions are as such:

CA1: Initial support for the applicability of VGT to automate manual scenario-based industrial test cases when performed by experts,

CA2: Initial support for the positive return on investment of the technique, and

CA3: Comparative results regarding the benefits and drawbacks of two VGT tools used in industrial practice.

This work also provided an industrial contribution to Saab with decision support regarding which VGT tool to adopt.

### 1.4.2 Paper B: Dynamic evaluation

Paper B, presented in Chapter 3, is titled "Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study". The paper presents a single, holistic, case study at the mission-critical military control system software development company Saab AB in the Swedish city of Järfälla.

***Research Objective:*** The objective of the study was to acquire support for the industrial applicability of VGT when adopted and used by practitioners. Additionally, to identify challenges, solutions, costs of adoption and use of VGT and finally the practitioners' perceptions about the technique.

***Methodology:*** The case study was performed in collaboration with Saab, where two testers used the open source VGT tool Sikuli to automate several manual test suites, at Saab referred to as acceptance test descriptions or ATDs, for the system in the continuation called the SUT. The SUT was mission critical, tested with 40 ATDs containing approximately 4000 use cases with a total execution time of 60 man weeks (2400 hours).

The study consisted of three phases, where Phase 1 was an exploratory pre-study performed to elicit the company's expectations on VGT, the company's test practices, SUT information, etc. In Phase 2, a four month project was conducted where the testers automated three out of the 40 ATDs with VGT, during which data collection was performed through telephone and e-mail due to the geographical distance between the researchers and the company. Finally in Phase 3, semi-structured interviews were used to verify the study's previous results and elicit the practitioners' perceptions about Sikuli and VGT.

***Results:*** The study's results were divided into three parts according to the study's phases, i.e. pre-study, case-study and post-study. ***The pre-study*** provided contextual information and showed that Saab needed VGT because of the high costs associated with the SUT's manual testing.

Further, ***the case study*** showed that the VGT scripts had been implemented as 1-to-1 mappings of the manual test cases and therefore consisted of small use cases, combined in meta-models, into longer test scenarios. This architecture was perceived beneficial to lower development and maintenance costs since the modular design facilitated change and reuse of use case scripts [40].

In addition, the VGT scripts executed 16 times faster than the manual tests, identified all regression defects the manual tests found but also defects that were previously unknown to the company. These additional defects were found by changing the order of the test scripts between executions that resulted in stimulation of previously untested system states. As such, VGT improved both the test frequency and the defect-finding ability of Saab's testing compared to their manual ATD testing.

However, six challenges were found with VGT; including high SUT-Script synchronization costs, high maintenance costs of older scripts or scripts written by other testers, low (70 percent) success-rate of Sikuli's image recognition when used over a virtual network connection (VNC) (100 percent locally), unstable Sikuli behavior, etc. These challenges were solved with ad hoc solutions; for instance, minimized use of VNC, script documentation, script coding standards, etc.

Additionally, three months into the study a large change was made to the SUT that required 90 percent of the VGT scripts to be maintained. This maintenance provided initial support for the feasibility of VGT script maintenance, measured to 25.8 percent of the development cost of the suite. The scripts' development costs were also used to estimate the time to positive ROI of automating all 40 ATDs, which showed that positive ROI could be achieved within 6-13 executions of the VGT test suite.

Finally, the ***post-study*** showed that VGT was perceived as both valuable and feasible by the testers, despite the observed challenges.

***Contributions:*** The main contributions provided by this study are as such:

CB1: Support that VGT is applicable in industrial practice when adopted and applied by practitioners in an industrial project environment,

CB2: Additional support that positive ROI can be achieved from adopting VGT in practice,

CB3: Initial support that the maintenance costs of VGT scripts can be feasible, and

CB4: Challenges and solutions related to the adoption and use of VGT in the studied project.

Consequently the study supports the contributions made by *Paper A* regarding the industrial applicability of VGT but also provides initial feasibility support for the technique. However, the study also reported several challenges with the technique that warranted further research.

### 1.4.3 Paper C: Challenges, problems and limitations

Paper C, presented in Chapter 4, is titled "Visual GUI Testing in Practice: Challenges, Problems and Limitations". The paper presents a multiple, holistic, case study with results from two cases. The first case was performed at the safety-critical air-traffic management software development company Saab AB in the Swedish city of Gothenburg, referred to as Case 1, whilst the second case was the case presented in *Paper B*, referred to as Case 2.

***Research Objective:*** The objective of the study was to identify challenges, problems and limitations (CPLs) associated with the adoption and use of VGT in practice and general solutions to these CPLs. A secondary objective was also to find support for the ROI of adopting VGT.

***Methodology:*** Both cases were performed independently of each other with engineers that transitioned manual test suites into VGT. This design allowed contextual CPLs and solutions to be distinguished from general CPLs and solutions. CPLs were primarily collected in Case 1, triangulated with results from Case 2, and systematically analyzed to determine the CPLs origin, generalizability, commonality to other CPLs, perceived impact, etc. These properties were then used to classify the CPLs into three tiers of varying abstraction.

In addition, metrics on development costs and execution time were acquired, which were used to estimate the time to positive ROI of adopting VGT in comparison to manual testing, in both cases.

***Results:*** 58 CPLs were identified in the study that were classified into three tiers with 26 unique CPL groups on the lowest tier (Tier 3), grouped into eight more abstract groups (Tier 2) and finally related to three top tier CPL root causes (Tier 1). Further, 34 out of the 58 CPLs related to the tested system, 14 to the test tool (Sikuli) and 10 to third party software (VNC and simulators). Hence, more than twice of the CPLs were associated with the SUT compared to the VGT tool.

The Tier 3 CPLs were also classified into six themes to analyze their impact on VGT, which, based on the study's results and related work, showed that the CPLs had varying impact but were also common to other automated and manual test techniques. This analysis also served to identify four general solutions to mitigate half of the study's identified CPLs. These solutions included systematic development of failure and exception handling, script documentation, minimized use of remote script execution over VNC and systematic SUT-script synchronization.

In addition, measured development costs were used to calculate the time to positive ROI of adopting VGT in the two cases, which showed that positive ROI would be achieved after 14 executions of the VGT suite in Case 1 and after 13 executions in Case 2. Additionally, based on the results and related work [16], a theoretical ROI cost model was created to serve for future research into the feasibility of VGT, which would, in addition to development costs, also take maintenance costs into consideration.

Finally, the study provided additional support for the defect finding ability of VGT since nine, in total, previously unknown defects were reported, three in Case 1 and six in Case 2.

***Contributions:*** The main contributions of this work are as such:

CC1: 29 unique groups of challenges, problems and limitations (CPLs) that affect the adoption or application of VGT in industrial practice,

CC2: Four general solutions that solve or mitigate roughly half of the identified CPLs, and

CC3: Results regarding the development costs, execution time, defect-finding ability and ROI of the use of VGT in industrial practice.

Additionally, the paper provides a general contribution to the body of knowledge of automated testing that currently only holds limited empirical support regarding CPLs [97].

### 1.4.4 Paper D: Maintenance and return on investment

Paper D, presented in Chapter 5, is titled "Maintenance of Automated Test Suites in Industry: An Empirical study on Visual GUI Testing". The paper presents a multiple, holistic, case study with results from the two companies Saab AB in Växsjö and Siemens Medical.

***Research Objective:*** The objective of the study was to evaluate the maintenance costs associated with VGT scripts. Hence, results essential to support the feasibility of the use of VGT in industrial practice.

***Methodology:*** The study was divided into two phases performed at Siemens Medical and Saab AB that were chosen through convenient sampling since companies that have used VGT for a longer period of time are rare.

Phase 1 was exploratory where three semi-structured interviews were performed to elicit practitioners' perceptions about VGT, VGT maintenance and experienced CPLs. At the time of the study, Siemens Medical had transitioned 100 out of 500 manual test cases into VGT with the VGT tool JAutomate [67] for the purpose of lowering the costs associated with manual scenario-based testing and to raise the quality of one of the company's systems.

In Phase 2, an empirical study was performed at Saab AB where a degraded [16] VGT suite was maintained by a researcher and a developer in two steps. First, the VGT suite was maintained for another version of the SUT which gave insights into the worst case maintenance costs of VGT suites. Second, the maintained VGT suite was migrated to a close variant to the SUT which gave insights into the costs of frequent maintenance of VGT suites. Fifteen (15) representative test scripts were maintained in total during the study, where representativeness was determined through analysis of the VGT suite and manual test specifications for the SUT. During the maintenance, measurements were taken on maintenance time per script, division of maintenance of images and script logic, number of found defects and script execution time. These measurements were then compared statistically to evaluate:

1. The difference in maintenance costs of frequent and infrequent maintenance.

2. The difference in maintenance costs of images and scripts, and

3. The difference between VGT development and maintenance costs,

Finally, the measurements were visualized in the theoretical cost model developed in *Paper C*.

***Results:*** Statistical analysis of the acquired measurements provided several insights into the costs associated with VGT script development and maintenance. First, the costs of frequent maintenance was found to be statistically significantly lower than infrequent maintenance. Second, the maintenance costs of logic and images are equal in degraded VGT test suites but image maintenance is significantly lower if the test suite is maintained frequently. Finally, the maintenance cost of a script, per iteration of maintenance, is lower than the development cost of the script, regardless of frequent or infrequent maintenance.

Further, eight defects were identified during the study, some during the maintenance effort and others by running the VGT scripts, which provides further support for the defect finding ability of VGT.

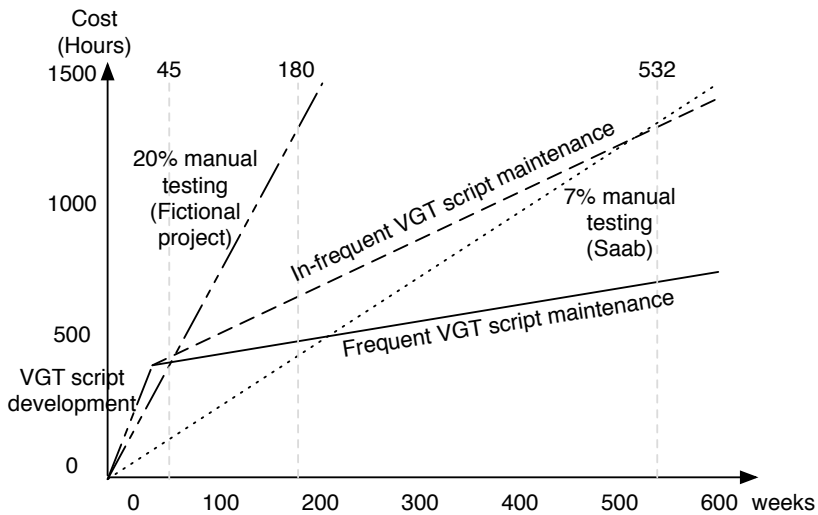


Figure 1.6: *Model of the measured development and maintenance costs of VGT compared to the costs of manual testing.*

Figure 1.6 presents a visualization of the time to positive ROI of VGT adoption and use compared to manual regression testing based on extrapolated cost data from the study. In the figure, VGT script development and frequent maintenance is shown with a ***solid line*** and VGT script development and infrequent maintenance with a ***long-dashed line***. Further, manual testing at Saab, which was seven percent of the total development costs of the project, which had 60 week iterations, is shown with a ***short-dashed line*** to be compared to a fictional project with 20 percent manual testing shown with a ***mixed long- and short-dashed line***. ***Gray, dashed, vertical lines*** show when positive ROI is reached in the different cases.

In the fictional project, positive ROI is reached in 45 weeks, i.e. within one development iteration of the project. However, in Saab AB's context, positive ROI would only be reached in 180 weeks if frequent maintenance was used and in an infeasible 532 weeks with infrequent maintenance. Hence, VGT

script maintenance is feasible but a VGT suite requires frequent maintenance if positive ROI is to be reached in a reasonable amount of time. Additionally, the time to positive ROI is dependent on the amount of manual testing performed in a project, i.e. in a project with more manual testing, positive ROI is reached faster. These results were supported by the interviews at Siemens Medical (Phase 1) where VGT script maintenance was associated with substantial cost and required effort, i.e. up to 60 percent of the time spent on VGT each week. However, despite these challenges, the technique was still considered beneficial, valuable and mostly feasibly by the practitioners.

***Contributions:*** The main contributions of this study are as such:

CD1: That maintenance of VGT scripts is feasible in practice, shown both through qualitative and quantitative results from two companies with two different VGT tools,

CD2: That maintenance of a VGT script:

(a) when performed frequently is significantly less costly than when performed infrequently,

(b) is significantly less costly per iteration of maintenance than development, and

(c) images are significantly less costly than maintenance of script logic.

CD3: That VGT scripts provide value to industrial practice, shown both with qualitative statements from practitioners and the technique's defect finding ability where eight defects were found using the technique, and

CD4: A ROI cost model based on actual data from industrial practice that verifies that this theoretical cost model is valid for automated test adoption and maintenance.

Hence, VGT can be feasible in practice but additional research was still warranted after the study to verify these results after long-term (years) use of VGT in practice.

### 1.4.5 Paper E: Long-term use

Paper E, presented in Chapter 6, is titled "On the Long-term Use of Visual GUI Testing in Industrial Practice: A Case Study". The paper presents a single, embedded, case study with results from the company Spotify.

***Research Objective:*** The objective of the study was to evaluate the long-term use of VGT in industrial practice, including short- and long-term challenges, problems and limitations (CPLs) and script maintenance costs. A secondary objective was to evaluate what alternative techniques that are used to VGT and to evaluate their benefits and drawbacks.

***Methodology:*** The study was divided into three steps where Step 1 was a pre-study, at Spotify, to acquire information about the company's use of VGT, willingness to participate in the study and what people to interview in the study (acquired through snowballing sampling [98]), etc.

In Step 2, four interviews were conducted with five employees at Spotify that had detailed knowledge about how VGT and alternative automated test

techniques were used at the company. Additionally, the interviews were complemented with two workshops, one exploratory in the beginning of the study and one with one person to verify previously collected results and to identify the company's future plans for VGT.

Finally, VGT was statistically compared to an alternative test technique developed by Spotify (the Test Interface) based on properties acquired in the interviews that were quantified based on the techniques' stated benefits and drawbacks.

***Results:*** VGT was adopted at Spotify after an attempt to embed interfaces for GUI testing (the Test interface) into the company's main application had failed due to lack of developer mandate and high costs. Further, because the application lacked the prerequisites of most other test automation frameworks, VGT became the only option. VGT was adopted with the tool Sikuli [54] and its success could be accounted to three factors:

1. The use of an incremental adoption process that began with a pilot project,

2. The use of best engineering practices to create scripts, and

3. The use of a dedicated adoption team.

Several benefits were observed with VGT, such as value in terms of found regression defects, robust script execution in terms of reported false test results, feasible script maintenance costs in most projects, support for testing of the release ready product, support for integration of external applications without code access into the tests, etc. Additionally, VGT integrated well with the open source model-based testing tool Graphwalker for model-based Visual GUI Testing (MBVGT). MBVGT made reuse and maintenance of scripts more cost-effective.

However, several drawbacks were also reported, such as costly maintenance of images in scripts, inability to test non-deterministic data from databases, limited applicability to run tests on mobile devices, etc. Because of these drawbacks, Spotify abandoned VGT in several projects in favor of the originally envisioned "Test interface" solution which became realizable after the adoption of VGT due to VGT's impact on the company's testing culture. Hence, VGT had shown the benefits of automation which gave developers mandate to adopt more automation and create the Test interface. These interfaces are instrumented by Graphwalker models that use the interfaces in the source code to collect state information from the application's GUI components that is then used to assert the application's behavior. This approach is beneficial since it notifies the developer if an interface is broken when the application is compiled, which ensures that the test suites are always maintained.

Additionally, the Test interface has several benefits over VGT, such as better support for certain test objectives (e.g. tests with non-deterministic data), faster and more robust test execution, etc. However, the Test interface also has drawbacks, such as inability to verify that the pictorial GUI conforms to the application's specification, inability to perform interactions equal to a human user, required manual synchronization between application and scripts, lack of support for audio output testing, etc.

Analysis based on quantification of these properties, i.e. benefits and drawbacks, also showed that there is significant difference between VGT and the Test interface. Hence, the techniques have divergent properties that make the more or less suitable in different contexts.

Finally, based on the study's results, results from *Papers B and C* and related work [99], a set of 14 guidelines were synthesized to provide practitioners with decision support and guidance to avoid pitfalls during adoption, use and long-term use of VGT in practice.

***Contributions:*** The main contributions of this paper are that:

CE1: VGT can be used long-term in industrial practice, as shown by Spotify's use of Sikuli for many years,

CE2: VGT has several benefits, including its flexible use that allows it to integrate external applications into the tests and test products ready for customer delivery,

CE3: VGT has many challenges, including robustness issues, possibly due to the immaturity of the technique's tools,

CE4: There are alternatives to VGT in practice with benefits such as higher robustness but with drawbacks that they do not verify that the pictorial GUI conforms to the system's specification, etc.

CE5: A set of 14 guidelines to support the adoption, use and long-term use of VGT in industrial practice.

The study thereby complements the missing results from *Paper D* regarding the long-term feasibility of VGT in practice.

### 1.4.6 Paper F: VGT-GUITAR

Paper F, presented in Chapter 7, is titled "Conceptualization and Evaluation of Component-based Testing Unified with Visual GUI Testing: an Empirical Study". The paper presents a combined study with an experiment and a case study performed in an academic setting.

***Research Objective:*** The objective of the study was to compare the differences in fault-finding ability of $2^{nd}$ and $3^{rd}$ generation (VGT) tools with respect to false test results for system and acceptance tests. A secondary objective was to combine the tool GUITAR with Sikuli into a hybrid tool called VGT-GUITAR and evaluate the two tools' capabilities on open source software.

**VGT-GUITAR:** VGT-GUITAR is an experimental tool that was developed based on the tool GUITAR's GUI ripping and MBT functionality [58], explained in Section 1.2.3. VGT-GUITAR extends GUITAR's ripper with bitmap ripping to acquire screenshots of the SUT's GUI components. These screenshots are then used during replay of test cases, generated by GUITAR, in a VGT driver (a Sikuli script) to interact with the SUT's pictorial GUI rather than by hooking into the SUT. For additional detail about the tool, see Chapter 7.

***Methodology:*** The study began with an experiment where a custom built GUI-based application was mutated using GUI mutation operators, defined

during the study, to create 18 faulty versions of the application. A test suite was then generated for the original version of the application that was executed with GUITAR and VGT-GUITAR (Independent variable) on each mutant to measure the number of correctly identified mutants, false positives and false negative test results (Dependent variables). The dependent variables were then analyzed to compare the two techniques in terms of reported false positives and negatives for system and acceptance tests, where system tests evaluated the SUT's behavior whilst acceptance tests also took the SUT's appearance into account. In addition the execution time of the scripts in the two tools were recorded and compared.

The study was concluded with a case study where GUITAR and VGT-GUITAR were applied on three open source applications to identify support for the tools' industrial applicability.

***Results:*** Statistical analysis of the experiment's results showed that $3^{rd}$ generation scripts report statistically significantly more false positives for system tests than $2^{nd}$ generation tools and that $2^{nd}$ generation tools report statistically significantly more false negative results for acceptance tests. These results could be explain by observations of the scripts' behavior on different mutants and relate to how the two techniques stimulate and assert the SUT's behavior, i.e. through hooks into the SUT or by image recognition. As an example, if the GUI's appearance was changed such that a human could still interact with it, e.g. by making a button larger, the $3^{rd}$ generation scripts would report a false positive result since the image recognition would fail. However, the $2^{nd}$ generation scripts would pass since the hook to the button still remained. In contrast, if the GUI's appearance was changed such that a human could not interact with it, e.g. by making a button invisible, the $2^{nd}$ generation scripts would produce a false negative since the hook allowed the script to interact with the invisible button. However, the $3^{rd}$ generation scripts would successfully fail because the image recognition would not find a match. The results of the experiment therefore indicate that a combination of the $2^{nd}$ and $3^{rd}$ generation techniques could be the most beneficial because of their complementary behavior for system and acceptance tests.

The proceeding case study did however show that VGT-GUITAR is not applicable in industrial practice since the tool had zero percent success rate on any of the open source applications, caused by technical limitations in the tool, e.g. in reality it could not capture screenshots of all GUI components. Additionally, the test cases were generated for GUITAR that can, for instance, interact with a menu item without expanding the menu, i.e. functionality that is not supported by $3^{rd}$ generation tools. Hence, further development is required to make VGT-GUITAR applicable in practice but the tool still shows proof-of-concept for fully automated $3^{rd}$ generation GUI-based testing due to its successful use for the simpler application in the experiment.

***Contributions:*** This study thereby provides the following main contributions:

CF1: Comparative results regarding the fault-finding ability of $2^{nd}$ and $3^{rd}$ generation GUI-based tools in terms of false test results for system and acceptance tests.

CF2: Initial support that a completely automated $3^{rd}$ generation test tool

could be developed even though the tool developed in the study, VGT-GUITAR, still requires additional work to become applicable in practice.

As such, the study primarily provides an academic contribution regarding future research directions for VGT but also results regarding the applicability of different GUI-based test technique's use for system and acceptance testing.

### 1.4.7 Paper G: Failure replication

Paper G, presented in Chapter 8, is titled "Replicating Rare Software Failures with Exploratory Visual GUI Testing". The paper presents an experience report provided by Saab AB in Gothenburg about how VGT was used to replicate and resolve a defect that had existed in one of the company's systems for many years. As such, unlike the previously included papers, this paper did not have any research objective or methodology and this section therefore only present a summary of the report.

***Experience report:*** The report presents how the company had received failure reports from their customers for several years regarding a defect in one of their systems that caused it to crash after long term use (3-7 months). These customer failure reports were however not sufficient to identify the defect and additional failure replication was therefore required. However, because the defect manifested so seldom in practice it was deemed too costly to resolve with manual practices, instead all customers were informed about the defect and were recommended to reboot the system with even frequency to mitigate failure.

In 2014 one of the company's developers found a way to remove the defect with a semi-automated test process that combined the principles of exploratory testing with VGT. In the process, a small VGT script was used to provide stimuli to the tested system's features in individual, mutually exclusive, components (methods). After each run, the script was modified by changing, or removing, methods that interacted with features that were perceived to not contribute to the manifestation of the defect, thereby incrementally ruling out which feature(s) caused the failure. Consequently an approach common to exploratory testing; simultaneous learning, test design and test execution [41,42]. The reason for the use of VGT for the process was because of the system's legacy that restricted the use of any other test automation framework.

By using the developed process, Saab AB was able to replicate the failure within 24 hours and resolve its defect within one calendar week. The defect was a small memory allocation leak, i.e. memory was not properly deallocated after a bitmap on the GUI had been rendered, but over time the leak built up to critical levels that caused the system to crash.

Post analysis of this case showed that the defect could have been found manually, at equal cost to the VGT approach but this defect analysis had to be performed by an engineer with specific technical knowledge about the system, which only a few developers at the company possessed. As such, this case shows that automated testing can provide value to a company in terms of quality gains rather than lowered costs.

***Contributions:*** The main contributions of this experience report are as such:

CG1: A success-story from industrial practice that shows that VGT can be used to replicate and resolve non-frequent and nondeterministic defects, and

CG2: A case where automated testing was paired with manual practices to create a novel, semi-automated, test practice, implying that similar processes can be achieved with other, already available, test frameworks in practice.

Additionally this case provides implicit support of the benefits of collaboration between academia and industry since it was academic transfer of VGT knowledge to Saab AB that resulted in the company's success story.

## 1.5 Contributions, implications and limitations

The objective of the thesis work was to find empirical evidence for, or against, the applicability and feasibility of VGT in industrial practice. In particular, what types of testing VGT can be used for, what the maintenance costs associated with VGT scripts are and what challenges, problems and limitations (CPLs) are associated with the short and long-term use of the technique in practice. Additionally, the research aimed to find ways to advance the industrial applicability of VGT and outline areas of future VGT research.

Evidence to fulfill this objective were collected through an incremental research process that included studies in academia and Swedish industry with several VGT tools and research methods. The individual contributions of these studies can be synthesized to answer this thesis research questions, as mapped in Table 1.4, which represent four key contributions:

1. Empirical evidence for the industrial applicability of Visual GUI Testing for different types of testing, i.e. regression, system, and acceptance testing of both deterministic and non-deterministic defects, and in different contexts, i.e. for daily continuous integration, for safety-critical and non-safety critical software.

2. Empirical evidence for the industrial feasibility of Visual GUI testing, including associated script maintenance costs and reasonable time to positive return on investment, given that frequent maintenance is used and a suitable amount of effort is spent on manual testing prior to VGT adoption.

3. Empirical evidence that there are challenges, problems and limitations associated with Visual GUI Testing that affect the adoption, short and long-term use of the technique in industrial practice, and

4. Technical and process solutions to advance Visual GUI Testing's industrial applicability, currently and in the future.

Together, these four contributions lets us draw the conclusion that VGT fulfills the industrial need for a flexible GUI-based test automation technique and is mature enough for widespread use in industrial practice. This conclusion is of particular value to companies that have GUI-based systems that

lack the prerequisites, e.g. specific technical interfaces, required by other test automation frameworks since VGT finally provides these companies with the means to automate tests to lower cost and raise software quality. However, there are still many challenges, problems and limitations (CPL) associated with VGT that are pitfalls that could prohibit the successful adoption, or longer term use, of the technique in practice. Pitfalls that must be taken into consideration by adopting companies and addressed, and mitigated, by future academic research.

The continuation of this section will present the detailed syntheses of the included research papers' individual contributions and how they provide support for the thesis objective and main conclusion.

### 1.5.1 Applicability of Visual GUI Testing in practice

VGT is first and foremost a test technique and its applicability must therefore be judged on its ability to find defects. Ample support for the defect-finding ability of VGT was provided in the thesis from *Papers B, C, D and indicated in E*, where it was reported that VGT identified all defects found by manual scenario-based regression testing but also new defects that practitioners said would not have been found without VGT. Hence, the technique can find defects with equal, or even greater, efficiency than manual, scenario-based, system testing. As such, VGT provides concrete value in practice and a suitable complement to existing test techniques, a conclusion also supported by explicit statements from practitioners in *Papers B, D and E*. Additionally, the experience report in *Paper G* shows that VGT can be used to find unknown, non-deterministic and infrequent defects.

Further, support for the thesis conclusions were acquired with several different VGT tools, i.e. Sikuli [54], JAutomate [67] and CommercialTool. Different benefits and drawbacks were identified with the tools but their core functionality, i.e. image recognition, make them equally applicable in practice. Additionally, image recognition is what provides VGT with its main benefit, its flexibility to test almost any GUI-driven system regardless of implementation language, operating system or even platform. This provides industrial practitioners with unprecedented ability to automate not only their SUT's but also the SUT's environment, e.g. simulators, external software, etc. Thereby allowing test cases that previously had to be performed manually to be automated, a statement supported by *Papers A, B, C and E*. However, these conclusions assume that the SUT has an accessible pictorial GUI, i.e. VGT has limited or no usability for systems that lack GUI's, such as server or general backend software.

| P. | ID | Contribution summary | RQ1 | RQ2 | RQ3 | RQ4 |
|---|---|---|---|---|---|---|
| A | CA1 | VGT is applicable for automation of manual scenario-based industrial test cases | X | | | |
| | CA2 | Initial support for the positive return on investment of VGT | | X | | |
| | CA3 | Comparative results on benefits and drawbacks of two VGT tools | X | | X | |
| B | CB1 | VGT applicable in an industrial project environment | X | | | |
| | CB2 | Positive ROI achievable after adoption of VGT in practice | | X | | |
| | CB3 | Initial support that the maintenance costs of VGT scripts can be feasible | | X | | |
| | CB4 | Challenges and solutions related to the adoption and use of VGT | | | X | |
| C | CC1 | 29 unique groups of challenges, problems and limitations (CPLs) that affect VGT | | | X | |
| | CC2 | Four general solutions that solve or mitigate roughly half of the identified CPLs | X | | | |
| | CC3 | Development costs, execution time, defect-finding ability and ROI of VGT | X | X | | |
| D | CD1 | Maintenance of VGT scripts is feasible in practice | | X | | |
| | CD2 | That maintenance of VGT (frequent, images, maintenance) is cost-effective | | X | | |
| | CD3 | VGT scripts provide value to industrial practice (e.g. finds defects) | X | | | |
| | CD4 | A ROI cost model based on data from industrial practice | | X | | |
| E | CE1 | VGT can be used long-term in industrial practice | X | X | | |
| | CE2 | VGT has several benefits in industrial practice | X | | | |
| | CE3 | VGT has many challenges | | | X | |
| | CE4 | There are alternatives to VGT in practice with some benefits over VGT | X | | | |
| | CE5 | 14 guidelines to support the adoption, use and long-term use of VGT in industrial practice | X | X | X | |
| F | CF1 | Comparative results regarding 2nd and 3rd generation GUI-based tools' abilities | X | | | X |
| | CF2 | Initial support for completely automated 3rd generation testing | | | | X |
| G | CG1 | VGT is able to replicate and resolve non-frequent and nondeterministic defects | X | | | |
| | CG2 | VGT can be paired with manual practices to create novel, semi-automated, test practices | | | | X |
| | | **Sum** | **12** | **9** | **5** | **3** |

Table 1.4: *Mapping of the individual contributions presented in Section 1.4 to the thesis research questions.* ***P*** *- Paper,* ***ID*** *- Identifier of contribution,* ***Cont.*** *- Contribution,* ***RQX*** *- Research question X,* ***CPLs*** *- Challenges, problems and limitations.*

Furthermore, in contrast to other automated test techniques, VGT enables regression testing of acceptance tests since, as discussed in Section 1.2.1, acceptance tests only differ from system tests in terms of the domain information embedded in the test scenarios, e.g. domain information that also includes the appearance of the SUT's GUI. Hence, since image recognition allows VGT to emulate a human user, it stands to reason that acceptance tests can also be automated, a conclusion also supported by *Papers B and F*. However, scenario-based scripts can only find defects in system states that are explicitly asserted, which, currently, delimits the use of VGT to automated acceptance regression testing. Acceptance testing otherwise requires cognitive reasoning and therefore a human oracle [69], which implies that it must be performed manually by an end user.

Additionally, VGT scripts execute faster than manual scenario-based test cases, reported in *Paper B* as much as 16 times faster than manual tests. However, more importantly, VGT scripts execute almost without cost and can therefore improve test frequency of GUI-based system tests, perceivably from weekly executions to daily execution or even per code commit. As a consequence, VGT can significantly improve the frequency of quality feedback to the system's developers and raise the system's quality.

However, VGT scripts are still slow in comparison to other automated test techniques, i.e. hundreds of automated unit tests can be executed at the same time as one VGT script. This conclusion presents a potential challenge for the use of VGT in practice, especially for continuous deployment where software, on commit, is automatically tested, delivered and deployed to the software's customer, discussed further in Section 1.5.2.

Further, whilst a unit test stimulates an individual software component, a single VGT script can stimulate all components of an entire sub-system, which thereby makes VGT an efficient means of achieving software component coverage. Additionally, a unit test only provides the tester with detailed knowledge of what component is broken, it does not provide information regarding what feature of the system is broken. In contrast, a VGT script can identify what feature of the system is broken but not in what component the defect resides. This observation implies that automation is required on several levels of system abstraction to provide test coverage of both software components and features of the software. In addition, it shows the value of VGT in practice since it is the only automated test technique that asserts the SUT's behavior through the same interface as the user.

However, VGT is not a replacement for existing techniques, e.g. manual regression testing, since, despite its defect-finding ability, it can only find defective SUT behavior that is explicitly asserted. In contrast, a human tester can observe faulty system behavior regardless of where or how it manifests on the GUI. However, as presented in *Papers D and E*, VGT can, and therefore only should, be used to mitigate costly, repetitive and error-prone manual testing, complemented with manual test practices, such as exploratory testing that finds new defects [42, 43].

As such, the results provided by this thesis show that VGT is applicable in industrial practice. A conclusion supported by results regarding the tools flexibility of use, improved test execution speed and defect-finding ability over manual testing. However, the technique is slower than other automated test

techniques, suffers from immature tooling and is suggested to report statistically significantly more false positives for system tests than $2^{nd}$ generation GUI-based testing. VGT should therefore be complemented with other automated test techniques to provide complete test coverage of a SUT, in particular in continuous delivery contexts.

### 1.5.2 Feasibility of Visual GUI Testing in practice

In order for VGT to be usable in practice it is not enough that it is applicable, it also needs to be feasible. Feasibility refers to the practical and cost-effective use of a technique over a longer period of time, which implies that the development and maintenance costs of scripts need to provide positive return on investment (ROI) compared to alternative testing practices, e.g. manual testing, over time.

VGT is best compared to manual regression testing because both techniques fulfill the same test objective and use the same types of inputs and outputs for SUT stimulation and assertion. Such a comparison is also valuable since manual GUI-based testing is the only available alternative for many companies [100], e.g. due to SUT legacy or other missing prerequisites for other automated test techniques.

However, feasibility also involves test execution time since VGT is primarily an automated regression testing technique which implies that VGT scripts should be executed frequently to provide fast feedback to developers, i.e. a practice that would be prohibited by too slow execution time.

Initial data on development costs and execution time of VGT scripts were acquired in *Papers A, B and C* and were used in the respective papers to calculate time to positive ROI. However, these results were acquired for different sized VGT suites and compared to varying manual test execution costs, which make them incomparable. Therefore the results were recalculated for the development, but not maintenance, of a fictional VGT suite of 100 test cases instead of 10, 300 and 33 test cases reported in *Papers A, B and C* respectively. These fictional development costs were then compared to the average *total time spent on manual testing per iteration* in the three cases (263 man-hours). Hence, in contrast to previous work where script development time was compared to the cost of running the manual test suites. This comparison thereby identifies how many times the VGT suite needs to be executed, after development, to equal the amount of manual testing that could have been performed for the same cost, i.e. the number of executions that are required for VGT adoption to provide positive ROI. Furthermore, the execution time of each fictional test suite was calculated that gives insights into the frequency with which the test suites can be executed in practice, i.e. hourly, daily or on only an even less frequent basis. The inputs and results of these calculations are presented in Table 1.5.

The table shows that the development costs of a VGT suite are considerable, i.e. in the order of hundreds of hours. However, once developed, the VGT suite provides positive ROI after 2.3 test suite executions, on average, compared to the total cost of manual testing with equivalent test cases.

Additionally, the table shows that the execution time of VGT suites can be significant and we therefore conclude that execution of a full VGT suite does

| Paper | Manual test costs | Script dev. time | Dev. time of 100 scripts | Script exe. time | Exe. time of 100 scripts | Positive ROI after |
|---|---|---|---|---|---|---|
| **A** | 150 mh | 195 min | 325 hours | 3 min 27 sec | 5 hours 45 min | 2 exe. |
| **B** | 488 mh | 206 min | 344 hours | 18-36 sec | 30-60 minutes | 2 exe. |
| **C** | 150 mh | 387 min | 645 hours | 27 min | 45 hours | 3 exe. |
| **Mean** | **263 mh** | **262 min** | **438 hours** | **10 min 4 sec** | **17 hours 15 min** | **2.3 exe** |

Table 1.5: *Table that summarizes the estimated results on development time, execution time and ROI of 100 VGT test cases from the results acquired in Papers A, B and C. Script development time is* ***compared to the average time spent on manual testing in the three projects, 263 hours****. In comparison, the calendar time spent in a six month project with 20 percent testing is 192 hours.* ***mh.*** *- Man-hours,* ***Dev.*** *- Development,* ***Exe.*** *- Execution,* ***ROI*** *- Return on investment,* ***Min*** *- Minutes,* ***Sec*** *- Seconds.*

not support faster than continuous integration on a daily basis. Therefore, test script prioritization is required to run VGT tests for regression testing and continuous integration on a hourly basis. However, in comparison to industrial state-of-practice of weekly manual regression tests, VGT still provides significantly improved test frequency [40].

Additionally, as can be seen in Table 1.5 the development costs and execution time for the VGT suite reported in *Paper C* was significantly higher than in the other two cases. The reason was because the test suite was developed to be robust, which was achieved by implementing several steps of failure mitigation code in the scripts and the test suite architecture. As such, we conclude that the architecture of VGT scripts play a role for the feasibility of VGT test development, which implies that there are VGT script best practices that should be followed, e.g. modularized test script design, scripts should be as short and linear as possible, etc. Further, more robust scripts take longer time to execute, which can stifle their use for continuous integration if the entire test suite needs to be executed often. As such there may exist a required tradeoff between robustness and execution time that needs to be taken into account during VGT script adoption and development.

However, the estimations presented in Table 1.5 do not take VGT script maintenance into account. VGT maintenance was evaluated explicitly in *Papers D and E*, where *Paper D* provided support for the feasibility of VGT script maintenance in two parts. First, the study showed, with statistical significance, that frequent maintenance of a VGT suite is less costly than infrequent maintenance. Additionally, maintenance per script per iteration of maintenance is lower than the development cost of a script, i.e. there is value in maintaining scripts rather than to rewrite them, and the cost of maintain-

ing images is lower than script logic. Second, the quantitative results were visualized in a ROI cost model, presented in Section 1.4.4 in Figure 1.6. These results indicate, in a best case, that the development and maintenance costs of a VGT suite provides positive ROI within on development iteration, given that at least 20 percent of the project's cost is associated with manual testing and that maintenance is performed frequently. However, if a company currently spends less time on manual testing and if scripts are maintained infrequently, the time to positive ROI could be several years, in Saab AB's case 532 weeks (or over 10 years).

Consequently, successful long-term use of VGT has several prerequisites. First, VGT needs to be integrated into the company's development and test process and the company's organization needs to be adopted to the changed process, e.g. to facilitate the need for frequent maintenance. Second, the developed VGT suite should follow engineering best practice, i.e. be based on a modularized architecture, have suitable amounts of failure mitigation, etc [40]. Further, test scripts shall be kept as short and linear as possible to mitigate script complexity, which is also mitigated by coding standards that improve script readability. Third, test automation should first, and foremost, be performed of stable test cases since this practice mitigates unnecessary maintenance costs and aligns with the techniques' primary purpose to perform regression testing. Additional factors were reported in *Paper D*, some that are common to other automated test techniques, but it is unknown how comprehensive this set of factors is and future research is therefore required to expand this set.

In summary we conclude that VGT is feasible in industrial practice with development and maintenance costs that are significant, yet manageable and provide positive return on investment. However, there are still challenges associated with the maintenance of VGT scripts that require suitable practices, organizational change as well as technical support, to be mitigated.

| # | Description | Affect | Impact | Support |
|---|---|---|---|---|
| 1 | VGT scripts (Sikuli) take over the computer during execution | Usage | Low | E |
| 2 | VGT documentation, guidelines and APIs, are lacking | Adoption | Low | B |
| 3 | Maintenance is affected by script readability, complexity, etc. | Maintenance | Low | B |
| 4 | 1-to-1 (manual-script) test cases are not always suitable | Maintenance | Medium | B,C |
| 5 | Manual test specifications don't always support scripting | Adoption | Medium | B,C |
| 6 | VGT tools (Sikuli and JAutomate) are immature/not robust | Usage | Medium | A,B,C,E |
| 7 | Image recognition is volatile, fails randomly | Maintenance | Medium | A,B,C |
| 8 | Script tuning (Synchronization, image similarity) is time consuming | Maintenance | Medium | B,C,E |
| 9 | SUT deficiencies (bugs, missing functionality) prohibit scripting | Adoption | High | B,C |
| 10 | Dynamic/non-deterministic output is a challenge for VGT scripts | Usage | High | A,E |
| 11 | VGT script image maintenance costs are significant | Maintenance | High | E |
| 12 | VGT scripts (Sikuli) have limited applicability for mobile testing | Usage | High | E |
| 13 | Remote script execution (VNC) negatively affects image recognition success rate | Adoption | High | B,C |

Table 1.6: *Summary of key reported CPLs. For each CPL, its affect and impact has been ranked. Affect refers to what the CPL affects (Adoption, Usage or Maintenance). Impact on how serious (Low, Medium or High) its presence is for a company. Column "Support" indicates in which studies the CPL was reported. The table is sorted based on impact.*

| Phase | # | Guideline | Description |
|---|---|---|---|
| **Adoption** | **1** | **Manage expectations** | It is not suitable/possible to automate anything and everything with VGT, consider what is automated and why? |
| | **2** | **Incremental adoption** | A staged adoption process that incrementally evaluates the value of VGT is suitable to minimize cost if the technique is found unsuitable. |
| | **3** | **Use a dedicated team** | A dedicated team can identify how/when/why to use VGT. |
| | **4** | **Use good engineering** | VGT costs depend on the architecture of tests/test suites and engineering best practices should therefore be used, e.g. modularization. |
| | **5** | **Consider used software** | Different software solutions, e.g. VGT tools and third party software, should be evaluated to find the best solution for the company's needs. |
| **Use** | **6** | **Change roles** | VGT can require new roles to be trained, which is associated with additional cost. |
| | **7** | **Development process** | VGT should be integrated into the development process, e.g. definition of done, and the SUT's build process, i.e. automatic execution. |
| | **8** | **Organization** | New roles require organizational changes that can disrupt development before the new ways of working settle. |
| | **9** | **Code conventions** | Code conventions help improve readability and maintainability of the scripts. |
| | **10** | **Minimize remote tests** | For distributed systems, VGT scripts should be run locally or use VGT tools with built in remote test execution support |
| **Long-term** | **11** | **Frequent maintenance** | The test process needs to prevent test cases degradation to keep VGT maintenance costs feasible long-term. |
| | **12** | **Measure** | The costs and value of VGT should be measured to identify improvement possibilities, e.g. new ways of writing scripts. |
| | **13** | **Version control scripts** | When the number of SUT variants grow, so do the test suites and they should therefore be version controlled to ensure SUT compatibility. |
| | **14** | **SUT life-cycle** | Positive return on investment of VGT adoption occurs after at least one iteration, so how long will the SUT live? |

Table 1.7: *Summary of guidelines to consider during the adoption, use or long-term use of VGT in industrial practice.*

### 1.5.3 Challenges, problems and limitations with Visual GUI Testing in practice

Sections 1.5.1 and 1.5.2 showed that VGT is applicable and feasible in industrial practice but also mentioned challenges, problems and limitations (CPLs) with the technique. These CPLs were primarily acquired in *Papers B, C and E* and have been summarized in Table 1.6. In the table, each CPL has been classified based on what phase of VGT it affects the most, i.e. *adoption*, e.g. adoption, implementation or creation of test scripts, *usage*, e.g. running the tests or using the tests for specific test purposes or SUTs and *maintenance*, e.g. script maintenance or long-term use of VGT in a project. Further, the impact of each CPL is classified as low, medium or high, where *low* means that it is an annoyance but requires little or no mediation, *medium* means that it has a negative effect that can be removed but requires mediation and *high* means that it has a negative effect but can only be mitigated, not removed, through mediation. Mediation, in turn, implies change to a company's procedures, organization, VGT scripts, SUT, etc.

Table 1.6 shows that many VGT CPLs relate to the technique's or its tools' immaturity, e.g. lack of robustness of both image recognition and the tools themselves. As the technology matures these should be less of a problem. Further, contextual factors, such as the test environment, seems to play an important role, e.g. scripting can be prohibited by poor manual test specifications, external applications or defects in the SUT. These observations indicate an interplay between many factors and it is therefore unlikely that any one, or a simple, solution can be found to solve all the CPLs. Instead, as discussed in Section 1.5.2, VGT requires process, organizational and technical changes to be applicable and feasible. Regardless, the results of this thesis show that most CPLs can be solved or mitigated, as presented in *Papers B and C*. Consequently, no CPL was identified that prohibits the technique's use in practice but several CPLs are considered more sever, e.g. the reported need for substantial image maintenance in *Paper E*.

To provide practitioners with support to avoid these CPLs and the pitfalls with the VGT, *Paper E* presented a set of 14 guidelines for the adoption, use and long-term use of VGT in industrial practice. These guidelines are based on best practices collected from all the studies presented in this thesis and have been summarized in Table 1.7. However, this set of guidelines is not comprehensive and future research is therefore required to expand this list.

Consequently, the results of this thesis show that there are many CPLs associated with VGT that must be considered by industrial practitioners during the adoption, use or maintenance of VGT or VGT scripts. However, these CPLs also provide an academic contribution regarding potential future research areas, i.e. future research to improve the technique's applicability and feasibility in practice.

### 1.5.4 Solutions to advance Visual GUI Testing

The conclusion that VGT is applicable and feasible in industrial practice opens up the possibility to also focus research on the advancement of the technique's use in practice. Advances that were studied in two of the thesis included

papers, i.e. *Papers F and G*.

In *Paper F*, initial research was performed towards fully automated VGT by creating a proof-of-concept tool, VGT-GUITAR. VGT-GUITAR was shown not to be applicable in practice due to technical limitations in the tool but the study outlines a foundation for flexible, automated, GUI-based, exploratory testing, i.e. an approach that could have considerable impact in practice to lower test related costs and improve software quality. Additionally, this approach could perceivably mitigate the development and image maintenance costs of VGT through automated acquisition of images from the SUT. Hence, a technical advancement that would improve the applicability and feasibility of the current VGT technique in practice.

Further, *Paper G* reported a novel test process for semi-automated fault replication with VGT. The process combines the practices of exploratory testing with stimuli provided by a simple VGT script and advances VGT by showing its applicability for finding infrequent and non-deterministic defects. In addition, the process provides companies with a means of improving their long-term test practices since long-term tests are generally performed over weeks or months in practice but generally without SUT stimuli. VGT could provide such stimuli on the same level of abstraction as a human user and thereby improve the representativeness of the test results for actual use of the SUT in practice.

These individual contributions imply that VGT is applicable for more than regression testing in practice, an observations that roots in its ability to emulate end-use behavior. Further, these results imply that VGT can be used in contexts when the expected output can not be acquired as an oracle, instead, a more basic oracle, e.g. a crash oracle, can be used together with a human oracle to find defects, as reported in *Paper G*. *Paper F* also indicates that expected outputs can be automatically acquired through GUI ripping but future work is required to develop and evaluate the theoretical foundation presented in *Paper F*.

In summary, solutions already exist to advance the applicability of VGT, e.g. by combining VGT with human oracles in semi-automated test practices or processes. Further, advances in tooling can, through future work, enable new and more advanced types of automated GUI-based testing based on VGT.

### 1.5.5 Implications

This thesis presents results with implications for both industrial practice and academia, e.g. decision support for practitioners and input for future academic research.

#### 1.5.5.1 Implications for practice

The main implication of this thesis is that VGT can be adopted and used in industrial practice, also over longer periods of time. This implies that companies with test (or software) related problems such as lacking interfaces required by other test frameworks, e.g. due to SUT legacy, or high test related costs, etc., now have a viable option for test automation. Additionally, adoption of VGT can help improve the test automation culture in a company, i.e. endorse

or mandate process, organizational or SUT changes that enable additional test automation in a company, as reported in *Paper E*.

For companies that have test automation, VGT provides a complement to their existing testing toolbox. In particular, VGT can provide high-level test automation for companies that currently test only on lower levels of system abstraction. However, VGT is not a replacement for manual GUI-based testing, instead it provides a suitable complement to mitigate the need for repetitive manual testing. This can reduce costs for the company by releasing resources, i.e. testers, which can instead focus on other types of testing, e.g. exploratory testing. Additionally, it may also raise the enjoyment of daily work for the individual, i.e. the human tester or developer. Statements supported by interview results from *Papers B, D and E*.

Further, because VGT scripts can run more frequently than manual tests, VGT can improve system quality [40]. This is an important implication of this work because it is not only of industrial benefit, but also of benefit to society, especially for safety-critical software systems, e.g. air-traffic management and medical systems, since improved quality can imply higher safety.

Another implication is that defect identification of infrequent and non-deterministic defects, e.g. defects that only manifest after longer periods of manual system interaction, are no longer out of scope due to cost. This result also implies that companies can improve their long-term test practices with continuous, user-emulated, stimuli that improve the tests' representativeness for use of the system in practice, as shown in *Paper G*.

Consequently, VGT provides several benefits to industrial practice by improving companies' test processes and thereby their software quality, improved software quality that benefits society as a whole.

#### 1.5.5.2 Future research

This thesis provides fundamental support to the body of knowledge on VGT regarding the technique's applicability and feasibility. As such, this research presents a stepping stone for future research to advance the technique and automated testing in practice. In this section we have divided future research into five different tracks; fundamental, technical, process, psychological and related research, and discuss how each track could be pursued.

***Fundamental*:** Fundamental future research on VGT regards additional support for the conclusions of this thesis, e.g. studies in more companies and contexts. The studies included in this thesis were performed with several companies, VGT tools and domains but more work is required to strengthen the body of knowledge of VGT and to ensure the generalizability of the results. Future work can also identify more CPLs and solutions to said CPLs as well as quantitative support for the long-term applicability of VGT. Hence, research in more types of contexts, companies and for other types of systems, e.g. web-systems, but also longitudinal research that follows the entire VGT life cycle from adoption to use to long-term use of the technique.

***Technical*:** Technical future research refers to technical advancement of the technique itself, its tools and the image recognition algorithms it is built on. These improvements could help mitigate the CPLs identified with VGT, e.g. the robustness of available VGT tools, image recognition algorithms as

well as costs associated with image maintenance.

In addition, this track refers to research into novel technical solutions based on VGT such as completely automated VGT, as outlined by *Paper F*. This research could also help solve the image maintenance CPL associated with VGT, for instance through GUI ripping which would also allow VGT scripts to easily be migrated between variants of a SUT, since, as reported in *Paper E*, test script logic can be reused between variants of an application but images cannot.

***Process*:** This track regards research into processes and practices to improve the adoption, use or maintenance of VGT, the importance of which discussed in *Papers D and E*. For instance, adoption of VGT should be performed incrementally, developed test scripts should be short and linear, the test suite should have a modularized architecture, etc. *Paper E* presented a set of best practice guidelines for VGT but additional work is required to create a more comprehensive set and to evaluate the current ones validity and impact in more companies and domains. This track also includes research into how VGT should be incorporated with other test techniques and practices in a company, e.g. when and how to use VGT to lower cost and raise software quality. As discussed in Section 1.5.1, VGT can test that a SUT's features are working correctly but not where in the code a defect is, and VGT therefore needs to be complemented with testing on lower levels of system abstraction, especially for continuous delivery and deployment [45]. However, how to efficiently combine VGT with other automated test techniques is still a subject of future research.

Consequently, this track primarily focuses on research into improving best practice guidelines for how VGT should be adopted and used in industrial practice and in different contexts. In addition, this track includes development of novel practices and processes, as presented in *Paper G*, i.e. processes that make use of VGT for semi-automated testing.

***Psychology*:** The thesis explored the enjoyment of using VGT in *Papers B and D* but enjoyment is only one factor that affect the use of a tool or technique in practice, other factors such as stress, motivation, etc. are also of interest. Hence, psychological factors that could affect practitioners' perception of the technique and which could potentially be improved through improved tooling or practices. This research could also help answer why VGT has only seen moderate adoption in practice so far and why adoption of VGT seems to facilitate adoption of additional automation in a company as reported in *Paper E*. Additionally, this research could provide further insights to guide the research in the aforementioned fundamental and process oriented research tracks, e.g. what factors to focus on to improve the technique's applicability or feasibility in practice.

***Related research*:** This track relates to generalization of the results of this thesis for other automated test techniques. For instance, this thesis reports several CPLs and solutions that are general to other automated test techniques. However, future research is required to analyze if said solutions are applicable for other test techniques in practice. Further, only a few solutions were reported for the CPLs, but it is possible that solutions that exist for other techniques can be migrated to solve VGT CPLs. Hence, cross-technical migration of solutions to common CPLs.

| Research question | Internal validity | External validity | Construct validity | Reliability/ conclusion validity |
|---|---|---|---|---|
| **RQ1: Applicability** | High | High | High | Moderate |
| **RQ2: Feasibility** | High | High | High | Moderate |
| **RQ3: CPLs** | Moderate | Moderate | High | Moderate |
| **RQ4: Future** | Moderate | Moderate | Moderate | High |

Table 1.8: *Summary of the threats to validity of the thesis results for each of the thesis research questions.* ***CPLs*** *- Challenges, problems and limitations.*

In addition, this thesis showed that it is possible to expand the applicability of VGT by combining it with manual practices and human oracles, a practice that is assumed to be generalizable to other automated test techniques and thereby warrants future research. Such research is of industrial interest, and importance, since it could theoretically allow companies to reuse existing tools and techniques for new purposes, thereby expanding their usefulness and improve the company's developed software.

Finally, this track includes research of how to extend or improve other test techniques with VGT capabilities, as outlined in *Paper F* where a hybrid tool for GUI-based testing was created. This research was performed with the $2^{nd}$ generation GUI-based tool GUITAR but similar development could be done for other tools, e.g. the commonly used tool Selenium [55], to allow these tools to also assert the SUT's behavior through the SUT's pictorial GUI. Thus providing the tools with a wider range of applicability in practice.

### 1.5.6 Threats and limitations of this research

This section presents an analysis of the threats to validity of the results and conclusions presented in this thesis. Threats to validity were analyzed based on the internal, external and construct validity as well as reliability/conclusion validity of the work [71]. A summary of the evaluated validity for each research question has been presented in Table 1.8 where validity is classified either as low, moderate or high.

#### 1.5.6.1 Internal validity

Internal validity refers to the cohesion and coherence of the results that support a research question. This was achieved in the thesis with an incremental research process where each study was based on the results, or gaps of knowledge, from previous studies. Additionally, as can be seen in Table 1.4, the papers included in this work provide multiple support for each research

question and also results that complement each other. For instance, the quantitative results regarding the feasibility of VGT (*Papers B and D* could be triangulated by statements from practitioners that had used the technique for a longer period of time (*Papers D and E).* Similar connections were found for results that support the applicability, for instance regarding the technique's defect finding ability, both for regression testing (*Papers B, C, D and E*) and for infrequent/non-deterministic defects, (*Paper F*). Therefore the internal validity of the conclusions for research questions 1 and 2 are considered high.

However, the internal validity of identified CPLs is only considerate moderate because different, unique, CPLs were identified in different studies. This observation implies that there could be additional CPLs that can emerge in other companies and domains.

Lastly, the internal validity regarding advances of VGT are also perceived to be moderate because these results were only provided from two studies, i.e. *Paper F and G*, which had specific focuses that are perceived narrow compared to the many possible advances to VGT as outlined in Section 1.5.5.2.

#### 1.5.6.2 External validity

External validity refers to the generalizability of the results or conclusions for other contexts and domains. The external validity of the conclusions regarding the applicability and feasibility of VGT are considered high because the results for these conclusions were acquired in different companies and with different VGT tools. Additionally, the research results come from both companies developing safety-critical software as well as agile companies developing non-safety-critical applications.

CPLs were acquired in the same contexts as the technique's applicability and feasibility but many of the reported CPLs were context dependent and it is unknown how comprehensive the set of identified CPLs are. Therefore the external validity for this research question is only considered moderate. However, the reported practitioner guidelines were triangulated with both studies on VGT in different contexts and domains as well as related research. As such the external validity of these guidelines is considered high, but more work is required to expand and evaluate this set of guidelines in the future.

Finally, for VGT advances, the external validity of the results are only considered moderate because the thesis only includes two studies, *Papers F and G*, which provide insights into explicit advances of the technique and it is therefore unknown how valuable advances in these areas would be for different industrial contexts and domains.

#### 1.5.6.3 Construct validity

Construct validity refers to the research context's ability to provide valid results to answer the study's research questions. Most of the studies presented in this thesis were conducted in industrial practice and they are therefore perceived to have provided results of high construct validity to research questions 1, 2 and 3. However, research question 4, had less industrial support, only *Paper G*, whilst the other main contributor to the question, *Paper F*, was performed as an experiment in an academic setting with software applications,

and tools, with limited representativeness for software in industrial practice. Therefore, the construct validity for question 4 is only considered moderate.

#### 1.5.6.4 Reliability/conclusion validity

Reliability/conclusion validity refers to the ability to replicate the study with the same results. The majority of the studies presented in this thesis were industrial case studies which implies that none of these studies can be replicated exactly. To mitigate this threat, the research methodology of each study has been presented in detail, a practice that is perceived to allow the validity of the studies to be judged without replication, triangulate the studies' results between the studies and replicate the studies in similar contexts. Further, an effort has been made in this thesis to outline the overall research process for the thesis work. However, due to the lack of replicability, of the majority of the studies, the overall reliability/conclusion validity of this research is considered moderate with the exception of the study presented in *Paper F* that presents an academic experiment that would be replicable by another researcher provided the study's research materials.

## 1.6 Thesis summary

The objective of this thesis was to find evidence for, or against, the industrial applicability and feasibility of Visual GUI Testing (VGT). This objective was motivated by the industrial need for a flexible technique to test software systems at high levels of system abstraction to alleviate the need for manual testing techniques that are often costly, tedious and error-prone.

The thesis work followed an incremental research methodology that began with exploratory studies to evaluate the applicability of VGT in practice. The research proceeded with studies to explain the challenges, problems and limitations (CPLs) and maintenance costs associated with the technique and was concluded with a study to verify the previously collected results and acquire evidence for the long-term use of VGT in industrial practice. Lastly, potential advances of VGT were evaluated that also outlined new areas of research and development for, or based on, VGT.

The results of these studies show that VGT is applicable and feasible in industrial practice, where applicability was supported by results on:

- Faster test execution speed and improved test frequency over manual testing,
- Equal or greater defect finding ability than manual test cases,
- Ability to identify infrequent and non-deterministic defects that are too costly to find manually,
- Ability to verify the conformance of a SUT's behavior and appearance through the SUT's pictorial GUI, and
- Flexibility of use for any system with a GUI regardless of implementation language, operating system or platform.

In turn, the feasibility of the technique was supported by results on:

- Positive return on investment (ROI) of VGT adoption if frequent maintenance is performed,
- Maintenance costs that per iteration of maintenance per script are significantly lower than the development cost per script,
- Script execution times that allow VGT to be used for daily continuous integration, development and delivery, which also contributes to the applicability of VGT, and
- Results from industry that show its feasible use over many months or even years.

However, acquisition of these results also uncovered many challenges, problems and limitations associated with the technique, which include, but were not limited to:

- Robustness problems associated with present-day VGT tools and the image recognition algorithms they use,
- Substantial costs associated with maintenance of images,
- Required, costly to achieve, synchronization between scripts and SUT execution, and
- Environmental factors that affect the adoption, use or maintenance of VGT scripts, but also
- 14 practitioner oriented guidelines that serve to ease the adoption and use of VGT in industrial practice.

However, none of the identified CPLs was perceived, in our studies, to prohibit the use of VGT in industrial practice.

Because of the identified support for VGT, advances to the technique itself were also evaluated. First, by combining VGT with automated GUI component ripping, model-based testing and test case generation a more fully automated VGT approach was outlined. However, only initial results were acquired but enough to warrant future research which could help mitigate the costs of development and need for image maintenance reported from industrial practice. Second, an experience report from industry reported the use of VGT in a semi-automated exploratory process, which provides a broader research contribution since it shows that automated tools and techniques can be combined with manual practices to cover additional test objectives.

In summary, this thesis shows that VGT is applicable and feasible in industrial practice with several benefits over manual testing. The thesis also provides cost information and CPLs that are pitfalls that industrial practitioners must consider to make an informed decisions about VGT adoption. As such, this work provides a clear contribution for a wider industrial adoption of Visual GUI Testing. In addition, the thesis advances the knowledge on GUI-based testing and outlines several important areas of future research.

Papers A – G not included in the online version.

# Bibliography


[1] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[2] M. Grechanik, Q. Xie, and C. Fu, "Maintaining and evolving GUI-directed test scripts," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on.* IEEE, 2009, pp. 408–418.

[3] ——, "Creating GUI testing tools using accessibility technologies," in *Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on.* IEEE, 2009, pp. 243–250.

[4] M. Finsterwalder, "Automating acceptance tests for GUI applications in an extreme programming environment," in *Proceedings of the 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering.* Citeseer, 2001, pp. 114–117.

[5] A. Leitner, I. Ciupa, B. Meyer, and M. Howard, "Reconciling manual and automated testing: The autotest experience," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on.* IEEE, 2007, pp. 261a–261a.

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

[7] E. Dustin, J. Rashka, and J. Paul, *Automated software testing: introduction, management, and performance.* Addison-Wesley Professional, 1999.

[8] A. Onoma, W. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, pp. 81–86, 1998.

[9] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *Software Engineering, IEEE Transactions on*, vol. 27, no. 10, pp. 929–948, 2001.

[10] M. Grechanik, Q. Xie, and C. Fu, "Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on.* IEEE, 2009, pp. 9–18.

[11] Y. Cheon and G. Leavens, "A simple and practical approach to unit testing: The JML and JUnit way," *ECOOP 2002Object-Oriented Programming*, pp. 1789–1901, 2006.

[12] E. Sjösten-Andersson and L. Pareto, "Costs and Benefits of Structure-aware Capture/Replay tools," *SERPS06*, p. 3, 2006.

[13] F. Zaraket, W. Masri, M. Adam, D. Hammoud, R. Hamzeh, R. Farhat, E. Khamissi, and J. Noujaim, "GUICOP: Specification-Based GUI Testing," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 747–751.

[14] M. Olan, "Unit testing: test early, test often," *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 319–328, 2003.

[15] E. Weyuker, "Testing component-based software: A cautionary tale," *Software, IEEE*, vol. 15, no. 5, pp. 54–59, 1998.

[16] S. Berner, R. Weber, and R. Keller, "Observations and lessons learned from automated testing," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 571–579.

[17] 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.

[18] R. Potter, *Triggers: GUIding automation with pixels to achieve data access*. University of Maryland, Center for Automation Research, Human/Computer Interaction Laboratory, 1992, pp. 361–382.

[19] L. Zettlemoyer and R. St Amant, "A visual medium for programmatic control of interactive applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*. ACM, 1999, pp. 199–206.

[20] T. Yeh, T. Chang, and R. Miller, "Sikuli: using GUI screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 183–192.

[21] B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proceedings of the 26th international conference on software engineering*. IEEE Computer Society, 2004, pp. 273–281.

[22] I. Sommerville, *Software engineering, 6th ed.* Addison-Wesley Professional, 2000.

[23] M. Huo, J. Verner, L. Zhu, and M. A. Babar, "Software quality and agile methods," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. IEEE, 2004, pp. 520–525.

[24] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.

[25] G. Myers, C. Sandler, and T. Badgett, *The art of software testing.* Wiley, 2011.

[26] I. Sommerville, "Software Engineering. International computer science series," 2004.

[27] D. Graham, "Requirements and testing: Seven missing-link myths," *Software, IEEE*, vol. 19, no. 5, pp. 15–17, 2002.

[28] M. Ellims, J. Bridges, and D. C. Ince, "The economics of unit testing," *Empirical Software Engineering*, vol. 11, no. 1, pp. 5–31, 2006.

[29] T. Ericson, A. Subotic, and S. Ursing, "TIM - A Test Improvement Model," *Software Testing Verification and Reliability*, vol. 7, no. 4, pp. 229–246, 1997.

[30] T. M. King, A. S. Ganti, and D. Froslie, "Enabling automated integration testing of cloud application services in virtualized environments," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research.* IBM Corp., 2011, pp. 120–132.

[31] C. Lowell and J. Stell-Smith, "Successful Automation of GUI Driven Acceptance Testing," in *Proceedings of the $4^{th}$ International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 03)*, Berlin, Heidelberg, 2003, pp. 331–333.

[32] E. Gamma and K. Beck, "JUnit: A cook's tour," *Java Report*, vol. 4, no. 5, pp. 27–38, 1999.

[33] L. Williams, G. Kudrjavets, and N. Nagappan, "On the effectiveness of unit test automation at Microsoft," in *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on.* IEEE, 2009, pp. 81–89.

[34] H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," *ACM Computing Surveys (CSUR)*, vol. 29, no. 4, pp. 366–427, 1997.

[35] J. Ryser and M. Glinz, "A scenario-based approach to validating and testing software systems using statecharts," in *Proc. 12th International Conference on Software and Systems Engineering and their Applications.* Citeseer, 1999.

[36] R. Binder, *Testing object-oriented systems: models, patterns, and tools.* Addison-Wesley Professional, 2000.

[37] B. Regnell and P. Runeson, "Combining scenario-based requirements with static verification and dynamic testing," in *Proceedings of the Fourth International Workshop on Requirements Engineering-Foundations for Software Quality (REFSQ98), Pisa, Italy.* Citeseer, 1998.

[38] R. Miller and C. Collins, "Acceptance testing," *Proc. XPUniverse*, 2001.

[39] Q. Yang, J. J. Li, and D. M. Weiss, "A survey of coverage-based testing tools," *The Computer Journal*, vol. 52, no. 5, pp. 589–597, 2009.

[40] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. Mantyla, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *Automation of Software Test (AST), 2012 7th International Workshop on*. IEEE, 2012, pp. 36–42.

[41] J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect detection efficiency: Test case based vs. exploratory testing," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*. IEEE, 2007, pp. 61–70.

[42] J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 2005, p. 10.

[43] W. Afzal, A. N. Ghazi, J. Itkonen, R. Torkar, A. Andrews, and K. Bhatti, "An experiment on the effectiveness and efficiency of exploratory testing," *Empirical Software Engineering*, pp. 1–35, 2014.

[44] P. Schipani, "End User Involvement in Exploratory Test Automation for Web Applications," Ph.D. dissertation, TU Delft, Delft University of Technology, 2011.

[45] H. Holmström-Olsson, H. Alahyari, and J. Bosch, "Climbing the" Stairway to Heaven"–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, pp. 392–399.

[46] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 50.

[47] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," *Software Engineering: A Practitioner's Approach*, 2001.

[48] K. Beck and C. Andres, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[49] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 476–485.

[50] E. Alégroth, R. Feldt, and L. Ryrholm, "Visual gui testing in practice: challenges, problemsand limitations," *Empirical Software Engineering*, vol. 20, no. 3, pp. 694–744, 2014.

[51] E. Alegroth, Z. Gao, R. Oliveira, and A. Memon, "Conceptualization and Evaluation of Component-based Testing Unified with Visual GUI Testing: an Empirical Study," in *Proceedings of the 8$^{th}$ IEEE International Conference on Software Testing, Verification and Validation (ICST 2015)*, Graz, 2015.

[52] E. Horowitz and Z. Singhera, "Graphical user interface testing," *Technical report Us C-C S-93-5*, vol. 4, no. 8, 1993.

[53] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for android," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 72–81.

[54] T. Chang, T. Yeh, and R. Miller, "GUI testing using computer vision," in *Proceedings of the 28th international conference on Human factors in computing systems*. ACM, 2010, pp. 1535–1544.

[55] A. Holmes and M. Kellogg, "Automating functional tests using Selenium," in *Agile Conference, 2006*. IEEE, 2006, pp. 6–pp.

[56] T. Lalwani, M. Garg, C. Burmaan, and A. Arora, *UFT/QTP Interview Unplugged: And I Thought I Knew UFT!*, 2nd ed. KnowledgeInbox, 2013.

[57] 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. IEEE, 2005, pp. 967–972.

[58] B. N. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "GUITAR: an innovative tool for automated testing of GUI-driven software," *Automated Software Engineering*, vol. 21, no. 1, pp. 65–105, 2014.

[59] I. K. El-Far and J. A. Whittaker, "Model-Based Software Testing," *Encyclopedia of Software Engineering*, 2001.

[60] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*. ACM, 2007, pp. 31–36.

[61] P. Fröhlich and J. Link, "Automated test case generation from dynamic models," *ECOOP 2000 Object-Oriented Programming*, pp. 472–491, 2000.

[62] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2007.

[63] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[64] E. Alégroth, "On the Industrial Applicability of Visual GUI Testing," Department of Computer Science and Engineering, Software Engineering (Chalmers), Chalmers University of Technology, Goteborg, Tech. Rep., 2013.

[65] E. Börjesson and R. Feldt, "Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 350–359.

[66] T.-H. Chang, "Using graphical representation of user interfaces as visual references," in *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*. ACM, 2011, pp. 27–30.

[67] E. Alégroth, R. Feldt, and H. Olsson, "JAutomate: a Tool for System- and Acceptance-test Automation," *ICST*, 2012.

[68] TestPlant. (2013, Feb.) eggPlant. [Online]. Available: http://www.testplant.com/

[69] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "A comprehensive survey of trends in oracles for software testing," Technical Report Research Memoranda CS-13-01, Department of Computer Science, University of Sheffield, Tech. Rep., 2013.

[70] R. Harrison and M. Wells, "A meta-analysis of multidisciplinary research," in *Conference on Empirical Assessment in Software Engineering (EASE)*, 2000, pp. 1–15.

[71] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[72] J. A. Maxwell, *Qualitative research design: An interactive approach*. Sage Publications, Incorporated, 2004.

[73] C. Wohlin, P. Runeson, and M. Höst, *Experimentation in software engineering: an introduction*. Springer Netherlands, 2000.

[74] M. Brydon-Miller, D. Greenwood, and P. Maguire, "Why action research?" *Action research*, vol. 1, no. 1, pp. 9–28, 2003.

[75] C. Robson, *Real world research: a resource for social scientists and practitioner-researchers*. Blackwell Oxford, 2002, vol. 2.

[76] M. Matell and J. Jacoby, "Is there an optimal number of alternatives for Likert scale items? I. Reliability and validity." *Educational and psychological measurement*, 1971.

[77] B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*. Transaction Publishers, 2009.

[78] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 557–572, 1999.

[79] R. Barbour and J. Kitzinger, *Developing focus group research: politics, theory and practice*. Sage Publications Limited, 1999.

[80] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, "A model for technology transfer in practice," *Software, IEEE*, vol. 23, no. 6, pp. 88–95, 2006.

[81] S. Kausar, S. Tariq, S. Riaz, and A. Khanum, "Guidelines for the selection of elicitation techniques," in *Emerging Technologies (ICET), 2010 6th International Conference on*. IEEE, 2010, pp. 265–269.

[82] R. R. Young, "Recommended requirements gathering practices," *CrossTalk*, vol. 15, no. 4, pp. 9–12, 2002.

[83] S. Shiba, "The Steps of KJ: Shiba Method," 1987.

[84] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, vol. 10, no. 3, pp. 311–341, 2005.

[85] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[86] F. J. Fowler Jr, *Survey research methods*. Sage publications, 2008.

[87] P. Berander and A. Andrews, "Requirements Prioritization," *Engineering and Managing Software Requirements*, 2005.

[88] A. Bowling, *Techniques of questionnaire design*. Open University Press, Maidenhead, UK, 2005.

[89] A. Bryman, "The debate about quantitative and qualitative research: a question of method or epistemology?" *British Journal of Sociology*, pp. 75–92, 1984.

[90] G. Wickström and T. Bendix, "The" Hawthorne effect"what did the original Hawthorne studies actually show?" *Scandinavian journal of work, environment & health*, pp. 363–367, 2000.

[91] V. Kampenes, T. Dybå, J. Hannay, and D. K Sjøberg, "A systematic review of quasi-experiments in software engineering," *Information and Software Technology*, vol. 51, no. 1, pp. 71–82, 2009.

[92] T. D. Cook, D. T. Campbell, and A. Day, *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston, 1979.

[93] L. Briand, K. El Emam, and S. Morasca, "On the application of measurement theory in software engineering," *Empirical Software Engineering*, vol. 1, no. 1, pp. 61–88, 1996.

[94] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *Software Engineering, IEEE Transactions on*, vol. 28, no. 8, pp. 721–734, 2002.

[95] W. H. Kruskal, "Historical notes on the Wilcoxon unpaired two-sample test," *Journal of the American Statistical Association*, vol. 52, no. 279, pp. 356–360, 1957.

[96] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *IEEE International Conference on Software Engineering (ICSE)*, 2011.

[97] D. Rafi, K. Moses, K. Petersen, and M. Mantyla, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *Automation of Software Test (AST), 2012 7th International Workshop on*, june 2012, pp. 36 –42.

[98] C. Kendall, L. R. Kerr, R. C. Gondim, G. L. Werneck, R. H. M. Macena, M. K. Pontes, L. G. Johnston, K. Sabin, and W. McFarland, "An empirical comparison of respondent-driven sampling, time location sampling, and snowball sampling for behavioral surveillance in men who have sex with men, Fortaleza, Brazil," *AIDS and Behavior*, vol. 12, no. 1, pp. 97–104, 2008.

[99] T. Hellmann, E. Moazzen, A. Sharma, M. Z. Akbar, J. Sillito, F. Maurer *et al.*, "An Exploratory Study of Automated GUI Testing: Goals, Issues, and Best Practices," 2014.

[100] D. Hoffman, "Cost benefits analysis of test automation," *STAR West*, vol. 99, 1999.

[101] P. Li, T. Huynh, M. Reformat, and J. Miller, "A practical approach to testing GUI systems," *Empirical Software Engineering*, vol. 12, no. 4, pp. 331–357, 2007.

[102] P. Hsia, D. Kung, and C. Sell, "Software requirements and acceptance testing," *Annals of software Engineering*, vol. 3, no. 1, pp. 291–317, 1997.

[103] P. Hsia, J. Gao, J. Samuel, D. Kung, Y. Toyoshima, and C. Chen, "Behavior-based acceptance testing of software systems: a formal scenario approach," in *Computer Software and Applications Conference, 1994. COMPSAC 94. Proceedings., Eighteenth Annual International.* IEEE, 1994, pp. 293–298.

[104] T. Graves, M. Harrold, J. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 10, no. 2, pp. 184–208, 2001.

[105] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, B. Helmkamp, and D. North, "The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends," *Pragmatic Bookshelf*, 2010.

[106] A. Adamoli, D. Zaparanuks, M. Jovic, and M. Hauswirth, "Automated GUI performance testing," *Software Quality Journal*, pp. 1–39, 2011.

[107] J. Andersson and G. Bache, "The video store revisited yet again: Adventures in GUI acceptance testing," *Extreme Programming and Agile Processes in Software Engineering*, pp. 1–10, 2004.

[108] M. Jovic, A. Adamoli, D. Zaparanuks, and M. Hauswirth, "Automating performance testing of interactive Java applications," in *Proceedings of the 5th Workshop on Automation of Software Test.* ACM, 2010, pp. 8–15.

[109] A. Memon, M. Pollack, and M. Soffa, "Hierarchical GUI test case generation using automated planning," *Software Engineering, IEEE Transactions on*, vol. 27, no. 2, pp. 144–155, 2001.

[110] P. Brooks and A. Memon, "Automated GUI testing guided by usage profiles," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering.* ACM, 2007, pp. 333–342.

[111] A. Memon, "An event-flow model of GUI-based applications for testing," *Software Testing, Verification and Reliability*, vol. 17, no. 3, pp. 137–157, 2007.

[112] T. Illes, A. Herrmann, B. Paech, and J. Rückert, "Criteria for Software Testing Tool Evaluation. A Task Oriented View," in *Proceedings of the 3rd World Congress for Software Quality*, vol. 2, 2005, pp. 213–222.

[113] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual network computing," *Internet Computing, IEEE*, vol. 2, no. 1, pp. 33–38, 1998.

[114] L. Fowler, J. Armarego, and M. Allen, "Case tools: Constructivism and its application to learning and usability of software engineering tools," *Computer Science Education*, vol. 11, no. 3, pp. 261–272, 2001.

[115] S. Eldh, H. Hansson, and S. Punnekkat, "Analysis of Mistakes as a Method to Improve Test Case Design," in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on.* IEEE, 2011, pp. 70–79.

[116] J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in *Empirical Software Engineering, 2005. 2005 International Symposium on*, nov. 2005, p. 10 pp.

[117] J. J. Gutiérrez, M. J. Escalona, M. Mejías, and J. Torres, "Generation of test cases from functional requirements. A survey," in *4ş Workshop on System Testing and Validation*, 2006.

[118] C. Cadar, P. Godefroid, S. Khurshid, C. S. Pasareanu, K. Sen, N. Tillmann, and W. Visser, "Symbolic execution for software testing in practice: preliminary assessment," in *Software Engineering (ICSE), 2011 33rd International Conference on.* IEEE, 2011, pp. 1066–1071.

[119] D. R. Hackner and A. M. Memon, "Test case generator for GUITAR," in *Companion of the 30th international conference on Software engineering*. ACM, 2008, pp. 959–960.

[120] V. Vizulis and E. Diebelis, "Self-Testing Approach and Testing Tools," *Datorzinātne un informācijas tehnoloģijas*, p. 27, 2012.

[121] E. Alégroth, R. Feldt, and H. Olsson, "Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study," *ICST*, 2012.

[122] A. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," in *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE)*, 2003, pp. 260–269.

[123] A. M. Memon and M. L. Soffa, "Regression testing of GUIs," in *ACM SIGSOFT Software Engineering Notes*, vol. 28. ACM, 2003, pp. 118–127.

[124] K. Li and M. Wu, *Effective GUI testing automation: Developing an automated GUI testing tool*. Sybex, 2004.

[125] smartbear. (2013, Feb.) TestComplete. [Online]. Available: http://smartbear.com/products/qa-tools/automated-testing-tools

[126] E. Börjesson, "Multi-Perspective Analysis of Software Development: a method and an Industrial Case Study," *CPL*, 2010.

[127] B. Beizer, *Software testing techniques*. Dreamtech Press, 2002.

[128] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[129] C. Ebert, "The impacts of software product management," *Journal of Systems and Software*, vol. 80, no. 6, pp. 850–861, 2007.

[130] C. Mongrédien, G. Lachapelle, and M. Cannon, "Testing GPS L5 acquisition and tracking algorithms using a hardware simulator," in *Proceedings of ION GNSS*, 2006, pp. 2901–2913.

[131] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Visual vs. DOM-Based Web Locators: An Empirical Study," in *Web Engineering*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8541, pp. 322–340.

[132] S. Wagner, "A model and sensitivity analysis of the quality economics of defect-detection techniques," in *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM, 2006, pp. 73–84.

[133] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*. IEEE, 2009, pp. 201–209.

[134] C. Liu, "Platform-independent and tool-neutral test descriptions for automated software testing," in *Proceedings of the 22nd international conference on Software engineering*. ACM, 2000, pp. 713–715.

[135] M. Fewster and D. Graham, *Software test automation: effective use of test execution tools*. ACM Press/Addison-Wesley Publishing Co., 1999.

[136] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 272–281.

[137] A. Kornecki and J. Zalewski, "Certification of software for real-time safety-critical systems: state of the art," *Innovations in Systems and Software Engineering*, vol. 5, no. 2, pp. 149–161, 2009.

[138] A. Höfer and W. F. Tichy, "Status of empirical research in software engineering," in *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer, 2007, pp. 10–19.

[139] E. Börjesson and R. Feldt, "Automated system testing using visual GUI testing tools: A comparative study in industry," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 350–359.

[140] A. Marchenko, P. Abrahamsson, and T. Ihme, "Long-term effects of test-driven development a case study," in *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2009, pp. 13–22.

[141] H. Kniberg and A. Ivarsson, "Scaling Agile@ Spotify," *online], UCVOF, ucvox. files. wordpress. com/2012/11/113617905-scaling-Agile-spotify-11. pdf*, 2012.

[142] N. Olsson and K. Karl. (2015) Graphwalker: The Open Source Model-Based Testing Tool. [Online]. Available: http://graphwalker.org/index

[143] J. Carver, "The use of grounded theory in empirical software engineering," in *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer, 2007, pp. 42–42.

[144] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2012, no. 14.

[145] M. Weinstein. (2002) TAMS Analyzer for Macintosh OS X: The native Open source, Macintosh Qualitative Research Tool. [Online]. Available: http://tamsys.sourceforge.net/

[146] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software engineering," *Empirical Software Engineering*, pp. 1–29, 2014.

[147] N. J. Nilsson, *Principles of artificial intelligence*. Tioga Publishing, 1980.

[148] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[149] E. Alégroth, "Random Visual GUI Testing: Proof of Concept," *Proceedings of the 25$^{th}$ International Conference on Software Engineering & Knowledge Engineering (SEKE 2013)*, pp. 178–184, 2013.

[150] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[151] B. N. Nguyen and A. Memon, "An Observe-Model-Exercise* Paradigm to Test Event-Driven Systems with Undetermined Input Spaces," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 216–234, 2014.

[152] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.

[153] P. R. Mateo, M. P. Usaola, and J. Offutt, "Mutation at system and functional levels," in *Proceedings of the 3$^{rd}$ IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICSTW 2010)*, Paris, France, 2010, pp. 110–119.

[154] L.-O. Damm, L. Lundberg, and C. Wohlin, "Faults slip through a concept for measuring the efficiency of the test process," *Software Process: Improvement and Practice*, vol. 11, no. 1, pp. 47–59, 2006.

[155] N. Nyman, "Using monkey test tools," *STQE–Software Testing and Quality Engineering Magazine*, 2000.

[156] L. C. Briand, Y. Labiche, and M. Shousha, "Stress testing real-time systems with genetic algorithms," in *Proceedings of the 2005 conference on Genetic and evolutionary computation.* ACM, 2005, pp. 1021–1028.

[157] V. T. Rokosz, "Long-term testing in a short-term world," *IEEE software*, vol. 20, no. 3, pp. 64–67, 2003.

[158] T. Arts, J. Hughes, J. Johansson, and U. Wiger, "Testing telecoms software with quviq quickcheck," in *Proceedings of the 2006 ACM SIGPLAN workshop on Erlang.* ACM, 2006, pp. 2–10.

[159] A. Zeller, "Isolating cause-effect chains from computer programs," in *Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering.* ACM, 2002, pp. 1–10.