

# CHALMERS



## Component-based Capture & Replay vs. Visual GUI Testing: an Empirical Comparison in Industry

*Master of Science Thesis in Software Engineering*

Anmar Khazal

Ármann David Sigurdsson

CHALMERS UNIVERSITY OF TECHNOLOGY  
Department of Computer Science & Engineering  
Gothenburg, Sweden, June 2014

The Authors grants Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Authors warrants that they are the authors of the Work and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Authors shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Authors warrants hereby that they have obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Component-based Capture & Replay vs. Visual GUI Testing: an Empirical Comparison in Industry

Anmar Khazal  
Ármann David Sigurdsson

© Anmar Khazal, June 2014.  
© Ármann David Sigurdsson, June 2014.

Examiner: Richard Berntsson Svensson  
Supervisor: Emil Alégroth

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

## Abstract

Graphical user interfaces (GUIs) are becoming an essential part of all software, which presents new challenges for high-level testing, i.e. GUI based system and acceptance testing. Currently, these tests are primarily performed through manual practices that are associated with problems such as high cost, tediousness and error proneness. These problems have been proposed to be solvable with automated testing techniques. However, support for automated high-level testing is still limited as Component-based Capture & Replay (CC&R), the most commonly used automated GUI based testing technique in industry today, suffers from various limitations that affect its usability, robustness and foremost maintainability, which leads to higher costs. However, Visual GUI Testing (VGT), a novel technique with promising characteristics such as high script robustness, has therefore been proposed as a more suitable technique in industrial practice than its predecessor CC&R. However, the body of knowledge regarding the VGT technique's applicability is limited in regards to the technique's maintenance cost and robustness.

This thesis will present an empirical comparison in industry between CC&R and VGT. The goal of the thesis is to compare the two techniques in an industrial context and to further bridge the gap in empirical knowledge concerning the VGT technique's long term applicability in industrial practice. The thesis work was conducted at CompanyX, which develops schedule and long term planning systems for the avionics industry and was conducted in two phases. The first phase was a pre-study with the goal of determining the industrial context of CompanyX. The second phase was an industrial study performed with a quasi-experimental design which compared and evaluated the two testing techniques, in terms of development cost, maintenance cost and robustness.

The results from this thesis work showed that there exists a statistically significant difference between the techniques in terms of development costs and robustness. However, the results showed that there was no statistical difference between the techniques' maintenance costs. Furthermore, both techniques were found to be applicable in industry and are powerful techniques for automated GUI based testing. However, the techniques have different benefits and drawbacks in different contexts, which indicates that a combination of the techniques would be the most beneficial. Further research is however required to verify this claim.



## Acknowledgements

I would like to thank my supervisor Emil Alégroth at Chalmers for always being available and helpful through numerous and almost endless phone calls. This work would have never been possible without his guidance. I also want to thank my co-author Ármann for being a rewarding sounding board and terrible fussball player throughout the thesis. Last but not least, I would like to thank my loving parents Saad and Ithar, my admirable sister Rana, my hilarious brother Hedir and my wonderful girlfriend Gabriella who have had faith in me and supported me through good times and bad.

**Anmar Khazal, Gothenburg 3/6/14**

First, I would like to thank my supervisor Emil Alégroth at Chalmers for his guidance and motivation throughout this thesis. Second, my thanks go to Anmar Khazal for his co-operation during this thesis work. Thirdly, I want to thank my father Sigurdur, my mother Sigurlaug, my brother Vilhelm and my sister Margrét for their endless support during my education and for being my role models. Last but not least, I want to thank my best Rannveig Ása for her love, faith and support while writing this thesis.

**Ármann David Sigurdsson, Gothenburg 3/6/14**



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Goal . . . . .	3
1.2	Purpose . . . . .	3
1.3	Scope . . . . .	3
1.4	Thesis outline . . . . .	3
<b>2</b>	<b>Literature review</b>	<b>5</b>
2.1	System and Acceptance test . . . . .	5
2.2	Manual GUI based testing . . . . .	6
2.3	Automated GUI based testing . . . . .	7
2.3.1	Coordinate-based Capture & Replay . . . . .	7
2.3.2	Component-based Capture & Replay . . . . .	8
2.3.3	Visual GUI Testing . . . . .	10
<b>3</b>	<b>Context description</b>	<b>13</b>
<b>4</b>	<b>Methodology</b>	<b>17</b>
4.1	Scoping . . . . .	17
4.2	Planning . . . . .	18
4.2.1	Context selection . . . . .	18
4.2.2	Hypothesis formulation . . . . .	19
4.2.3	Variables selection . . . . .	20
4.2.4	Selection of subjects . . . . .	21
4.2.5	Experimental design . . . . .	22
4.2.6	Instrumentation . . . . .	23
4.3	Operation . . . . .	23
4.3.1	Preparation . . . . .	24
4.3.2	Execution . . . . .	24
4.3.3	Data validation . . . . .	26

<b>5</b>	<b>Results</b>	<b>27</b>
5.1	Development . . . . .	27
5.2	Maintenance . . . . .	32
5.3	Robustness . . . . .	34
<b>6</b>	<b>Discussion</b>	<b>38</b>
6.1	Development . . . . .	38
6.2	Maintenance . . . . .	40
6.3	Robustness . . . . .	40
6.4	General . . . . .	41
6.5	Tool discussion . . . . .	42
<b>7</b>	<b>Validity evaluation</b>	<b>44</b>
7.1	Internal validity . . . . .	44
7.2	External validity . . . . .	45
7.3	Construct validity . . . . .	45
7.4	Conclusion validity . . . . .	46
<b>8</b>	<b>Conclusion</b>	<b>47</b>
	<b>Bibliography</b>	<b>51</b>



# 1

## Introduction

**G**RAPHICAL USER INTERFACES (GUIs) are becoming the primary means of interacting with software systems [1]. The GUI handles user system interactions, such as mouse clicks, mouse movements and menu selections. The GUI then interacts with the underlying code with input events via messages or method invocations and because the GUI can be complex, it can take up a large part of the total system code, as much as up to 60% [1]. With this increased use and importance of GUIs, testing of system correctness through the GUI has become an essential part of the software verification and validation (V&V) process to ensure software quality [1, 2]. However, at the same time market demands for faster time-to-market delivery and higher software quality are increasing, which presents a concern for companies that rely on manual V&V processes because manual GUI based testing is considered time-consuming, tedious and error-prone [3, 4]. Automated testing lowers the required manual effort on test execution and the tests can be executed faster and more frequently than tests that are executed manually [5].

The most common automated GUI based testing technique in industry today is called Widget/Component-based Capture & Replay [6, 7], in this thesis referred to as Component-based Capture and Replay (CC&R). The CC&R technique makes use of widget/component properties to interact with the system under test (SUT). Hence, the CC&R technique operates on the GUI component level and therefore requires access to the underlying code of the SUT [3, 4, 5, 6, 7, 8]. As a consequence, CC&R tools interact with the SUT through direct GUI component invocation that differs from end user system usage where SUT interaction is performed through mouse and keyboard events instrumented by the operating system. Direct GUI component invocation makes the technique robust to GUI layout change as long as the SUT's GUI component API and the structure of component properties, e.g. labels and/or ID numbers, remain unchanged [9]. However, this property also limits CC&R tools in terms of usability, i.e. they are only applicable for applications written in certain programming languages, are

platform dependent and do not work for distributed systems. Furthermore, despite their robustness to change, CC&R test scripts are still subject to high maintenance costs due to required technical knowledge required by the script developer, i.e. knowledge about the SUT's inner structure and available GUI component properties [2, 10, 11, 12, 13, 14]. Due to CC&R's limitations, despite the technique's industrial use, we stipulate that further research and new techniques are required to meet the industrial demand for automated GUI based testing to replace or complement the currently used costly, tedious and error-prone manual GUI based testing practices [6].

Visual GUI Testing (VGT) [7] is a novel automated GUI based testing technique and a successor to CC&R with promising characteristics, e.g. high script robustness and usability, which could make it a suitable technique to fulfil industrial needs [9]. Unlike its predecessor, VGT mimics user interactions with the SUT by using image recognition to locate and interact with the SUT's GUI components as shown to the user on the computer monitor. Hence, the VGT technique operates solely on the SUT's GUI bitmap level by comparing expected bitmap graphics to the SUT's runtime bitmap graphics [3, 6, 7, 9]. Consequently making the technique blackbox and independent of SUT implementation, operating system or even platform since no SUT code level access or knowledge is required for use. The technique's approach of using image recognition also makes VGT scripts robust to GUI layout change since expected graphics can be found anywhere on the monitor. For the same reason, VGT scripts are also not affected by API code structure changes since the technique does not require access to the SUT's underlying code [3, 6, 7, 9]. The VGT technique is a good candidate to solve the problems that are presented for CC&R, because the VGT technique is perceived to be cheaper in terms of maintenance since less technical knowledge is required by the test developer [6, 7].

However, even though studies by Börjesson and Feldt [3] and Alégroth et al. [7] showcase the technique's industrial applicability there are still gaps in VGT's body of knowledge regarding the applicability and long term applicability of the technique. Additionally, to the authors' best knowledge, there are no studies that compare VGT to its predecessor CC&R in an industrial context, perceivably contributing to the sparse use of VGT in industrial practice [3, 15].

To bridge these gaps in knowledge, this thesis presents an empirical study where two instances of CC&R and VGT were compared in an industrial context. The comparison was performed through a quasi-experiment at CompanyX where quantitative data regarding the techniques' development time, maintenance time and robustness were gathered and analyzed statistically to answer the study's research questions. Research questions that encompassed which technique was perceived to be more beneficial in terms of the above stated metrics, e.g. development time, in the studied context.

Results of the study showed that a statistically significant difference exist between the techniques in terms of development costs and robustness, but regarding maintenance cost there was no statistical difference between the two techniques. Furthermore, the results of this study show that the two techniques have different properties that make them more suitable in different contexts and for different levels of test abstraction.

## 1.1 Goal

The main goal of this thesis is to compare the mature CC&R technique and the novel VGT technique in terms of the metrics; development and maintenance cost measured as time and robustness, in order to determine which technique has the better applicability for each metric. In order to reach this goal, comparative data gathered will be gathered and analysed in a quasi-experiment, conducted in an industrial context. Moreover, our sub-goal is to further bridge the empirical gap in knowledge concerning the VGT technique's long term applicability in the industrial context. In addition, our meta-goal is to determine if it is beneficial, in terms of the investigated attributes, for CompanyX to change their current GUI based testing technique from CC&R to VGT.

## 1.2 Purpose

Hence, the purpose of this thesis is three-fold, (1) to compare the CC&R technique and the VGT technique in an industrial context, (2) to find empirical evidence regarding the costs associated with VGT and thereby contributing to the body of knowledge regarding the technique's long term applicability in industrial practice, (3) to provide a more general contribution to the area of automated testing through the novel comparative industrial study of the CC&R and VGT technique.

## 1.3 Scope

The scope of this thesis work is limited to comparing the techniques CC&R and VGT by using only two tools within an industrial context. Furthermore, the open source VGT tool Sikuli was considered to be included in the study but because of time constraints it was discarded.

In addition, the outcome of this thesis work is perceived to only be generalizable for similar contexts as the one at CompanyX. Thus, because of the limited scope, this work is not considered holistic and further research is required to verify the findings presented in this thesis for other tools, companies and contexts. However, despite the limited scope, the study provides a general contribution to the body of knowledge of automated testing and a particular contribution to the limited body of knowledge of VGT.

## 1.4 Thesis outline

The Introduction section is followed by the Literature review section 2, which describes different levels of testing and the different GUI based testing techniques. The Literature review section also describes the tools that were used to represent the techniques in this study and it is followed by the Methodology section 4. In the Methodology section the company where the study was conducted is described as well as the research methods that were used to acquire the thesis work results. The results are presented in the

Results section 5 together with an analysis of said results. The analyzed results are then discussed and elaborated on in the Discussion section 6. Thereafter the threats to the result's validity is presented in the Validity evaluation section 7. Finally the thesis work is concluded in Conclusion section 8.

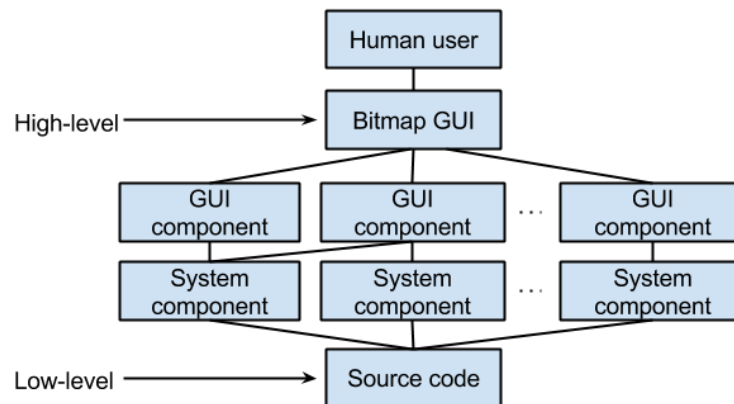
# 2

## Literature review

GUIs ARE BECOMING the primary method of interacting with software systems and developers are dedicating a large amount of software code into GUI implementation [1]. GUIs can constitute as much as 60% of the total software code [1] and therefore a common practice has become to perform GUI based system tests in order to ensure the correct operation of the overall software system [2].

### 2.1 System and Acceptance test

Testing of a software system is often done at different phases in the development process and testing at each phase serves a different purpose citealegroth2013industrial. In the early phases of a development process, testing is mostly done through unit testing of individual components [9]. Unit testing is a process of testing individual components, subprograms or procedures in the system, with the purpose to compare the actual output of a component, subprogram or procedure to an expected output [16]. Unit tests interact with the SUT directly through the source code and are therefore said to be performed on the lowest-level of a system abstraction [9], see Figure 2.1. In contrast, tests that are performed on the highest-level of the system abstraction, interact with the SUT through the GUI, e.g. system and acceptance tests [9]. Although, it is not always the case that system tests interact with a GUI they can also be performed on the system component level [9], see Figure 2.1. The purpose of a system test is to verify that all the components of the SUT work correctly together and that the SUT complies to the system's requirements [9, 16]. A requirement is a specification of a functionality or a condition that the system must possess or meet in order to satisfy the customer's or end-user's needs [17]. However, system tests do not assess if the SUT actually complies to the customer's or end-user' needs [16]. Compliance to customer or end-user needs are instead validated in acceptance tests [16].



**Figure 2.1:** Visualization of a general model of the abstraction levels of a system.

Currently, companies often use manual practices for GUI based system and acceptance tests, which are associated with tedious, time-consuming and error prone practices [9]. To mitigate the issues with these manual practices, test automation has been proposed as a solution [9]. However, most automated tests are performed on the lower levels of system abstraction, e.g. unit tests, and therefore perceived unsuitable for high-level tests, e.g. system and acceptance test [9, 18]. The reason is because a test on one level of system abstraction has different goals and different test logic than a test on another level of abstraction [18]. Unit tests for system tests are difficult to create because the unit tests have to include both the logical and chronological behavior of the individual components that the system tests aim to verify [18].

Nonetheless, automation techniques for GUI based tests exists, and will be presented in the following section with further discussion on the difference of manual and automated GUI based testing. Furthermore, different techniques to develop automated GUI based test scripts will be presented. In particular, the text will elaborate on the techniques that were used during this thesis work.

## 2.2 Manual GUI based testing

Manual GUI based testing is a commonly used approach for testing and quality assurance in software industry [6]. One type of manual testing is scenario-based manual testing, which is performed by a tester or developer that interacts with the SUT's GUI, i.e. by clicking on buttons or components on the screen, in a scenario predefined in a test case specification [10]. After the tester has performed an, or several, interaction(s) with the GUI component(s), (s)he manually verifies that the outcome on the screen is correct according to an expected result [10]. Each test case specification can contain several

scenarios and the scenarios can vary in length [9]. When the scenarios are long, the test case execution will take a long time to perform manually and will become tedious, which can lead to mistakes during the test execution [9].

Furthermore, since software systems are prone to change, they should be regression tested. Regression testing is a testing practice that is performed after a modification has been done to the system and is performed by rerunning all, or a set, of test cases to determine if the modifications have affected other previous working parts/functions of the SUT [16]. Manual GUI based regression testing can be associated with significant cost, especially if many test cases need to be rerun [10, 19]. Therefore, the process of manual regression testing is tedious, time consuming and can account for as much as 50-60% of the total software development cost [10, 19]. Hence, by automating the GUI-based test cases it would be possible to execute the tests quicker and more frequently than running the tests manually and thereby reduce the test effort and development cost of the SUT [5]. However, as previously mentioned, the techniques that support high level test automation have limitations that adversely affect their applicability.

## 2.3 Automated GUI based testing

Automated GUI based testing is tool-driven by tools where test cases are defined as scripts that are implemented using a scripting language, e.g. VBScript<sup>1</sup> [5]. These scripts define scenarios of actions that when executed, result in interactions being performed on the SUT's GUI, i.e. automatic input to the SUT [5]. Furthermore, these test scripts assert whether the SUT executed as intended, i.e. reported correct GUI output, and report any assertion violations as a test execution failure [5, 20]. These test scripts significantly reduce manual execution effort, because once the test cases are developed they can easily be re-executed [5, 20]. However, the scripts are still associated with a development cost, which for a large test suite can be substantial, especially in tools where the scripts need to be written manually [20]. To mitigate these costs, many tools for automated GUI based testing also support script recording [20], a practice that will be described in detail in the following subsection.

### 2.3.1 Coordinate-based Capture & Replay

The first generation of automated GUI based testing techniques relies completely on coordinates for recording and playback of test scenarios and is therefore referred to as Coordinate-based Capture & Replay (C&R) [6, 9]. The technique can be described in two steps. In the first step the user's manual interactions with the GUI, such as keyboard strokes, clicks at a set of coordinates, etc., are recorded in a test script. In the second step the recorded test script is replayed to automatically interact with the SUT, thereby automatically re-executing the scenario [1, 3, 6, 7, 9]. Thus automatically asserting the system's correctness, i.e. automated verification that the system behaves according to its requirements specification [1, 6]. However, the Coordinate-based Capture & Replay

---

<sup>1</sup><http://www.visualbasicscript.com/>

technique's scripts are fragile to GUI layout change [7, 15]. That is, if changes are made in the GUI layout after recording, the recorded coordinates may no longer be valid. This may in turn cause the script to fail when executed. Thus, the C&R technique suffers from limitations that affect the technique's usability, cost, robustness and maintainability [2, 10, 11, 12, 13, 14].

However, despite the techniques fragility to GUI layout change, the technique's scripts are perceived as robust to changes in the SUT's code or API, given that the changes do not affect the GUI [6, 7, 15]. This is because the technique operates mainly on the GUI bitmap level, i.e. test scripts are carried out by replaying interactions at exact coordinates on the monitor [6, 7].

### 2.3.2 Component-based Capture & Replay

Scripts developed in the second generation of automated GUI based testing techniques, here referred to as Component-based Capture & Replay (CC&R), are robust against GUI layout changes but fragile to API or code changes that affect the GUI components properties. The reason is because most CC&R tools acquire said GUI component properties through direct access to the source code of the SUT. Thus, the CC&R technique operates solely on the GUI component level. [3, 4, 5, 6, 7, 8]. To exemplify, during recordings of manual user interactions, the technique captures properties such as id, type or methods of GUI components. These GUI components are typically buttons, text fields or images. The captured properties are later used during playback of the test script to identify and interact with the SUT's components. For example, a component can be identified and located through its unique id property and/or unique name. Furthermore, interaction with a component can be performed by invoking the component's methods, such as a buttonclick [4, 6].

However, in order to assert the correctness of a SUT during script playback, verification points needs to be added to the test scripts that include the state of a set of SUT GUI component's properties after a set of inputs have been given [21]. For this reason, the verification points can be compared to snapshots of the SUT's state at a certain point in time. The verification points are then used to verify that the SUT reaches the same expected state during playback of a scenario by comparing the expected state with the actual state [9].

The usage of component properties is therefore perceived to make the CC&R technique more flexible and robust than the C&R technique when it comes to GUI layout changes [4]. However, CC&R's main drawback is that the component interactions makes the technique intrusive, i.e. it requires access to component property information and underlying source code of the SUT and therefore treats the SUT as a white-box [9]. Because of this property, most CC&R tools are limited to work with a certain set of programming languages or GUI libraries, i.e. no tool supports all programming languages even though there are tools that use the Windows operating system environment to interact with any windows component regardless of implementation, e.g. TestComplete [6].



Consequently, the CC&R technique suffers from limitations that affect the technique's usability, cost, robustness and foremost maintainability [2, 10, 11, 12, 13, 14]. Between 30 to 70 percent of CC&R test scripts require maintenance even after minor changes to the SUT, i.e. code or GUI layout changes [11, 12, 18], which affects the CC&R technique's applicability negatively and increases the amount of maintenance the test scripts require [8, 11, 12, 18].

Additionally, the CC&R technique does not interact with the GUI like a human tester manually would [9], e.g. instead of clicking on a button on the GUI through the operating systems mouse click functionality the tools invoke the click through the button's click method. Another example of the difference is when GUI components have been defined that are not set to be visible on the monitor. When this is the case, the technique can still identify the invisible components and interact with them, which would be impossible for a human [6, 13]. Consequently, such cases may cause inaccurate assertions results, i.e. false-negative results or false-positive results. Hence, resulting in verification points failing although no defect exists and verification points passing although a defect exists respectively.

Although the CC&R technique possesses beneficial properties that support its use for GUI based testing in practice, it has limited applicability for distributed software systems, cloud-based systems with limited access to the back-end and systems that are written in several programming languages. Furthermore, as mentioned, it is limited to testing systems written in the limited set of languages that the used CC&R tool supports. Due to CC&R's limitations further research into GUI test automation and new techniques is needed, despite the fact that CC&R's limitations are context dependant [6].

### **QuickTest Professional**

As mentioned, CC&R is a tool-driven technique and the CC&R tool that was used in this thesis work is Hewlett Packard's Quick Test Professional (QTP) version 11.00 [22, 23]. QTP is capable of recording user interactions on both high and lower levels of system abstraction, see Figure 2.1. To clarify, lower level recordings are performed through component level recordings. The high level recordings are performed at GUI bitmap level by using exact coordinates on the screen, i.e. the C&R technique. During low level recordings, QTP only captures information about windows and their components, such as text fields and buttons. This feature is called Object Recognition and the objects with their respective properties are stored in an object repository. The object repository allows the user to reuse saved objects and thus lower the development time of new test scripts. QTP also allows the user to choose which components should be stored. For example, the user can choose to only store the components that the user interacted with, all components in a certain window or component of a specific type. In contrast, during high level recordings the tool records all mouse movements, clicks and keyboard inputs instead. This is advantageous when the user wants to interact with an object that is not well defined, e.g. flash animations and drawing tools. QTP also allows users to write and execute automated GUI tests using a scripting language, i.e. Visual Basic

Script. The tool can also record a test log when requested. The scripts may be set up to perform an action repeatedly, e.g. to iterate over sets of data, such as spreadsheets. Furthermore, assertions in the tool are called Checkpoints and determine if a test passes or fails. Different types of checkpoints compare different types of data, e.g. text, object status and even bitmaps can be asserted if they exist in a particular place on the screen with image recognition. Furthermore, QTP allows the user to create and share an Action among test scripts, where an action is comparable to a function. Hence, actions contain code and allow reuse of code [22, 23].

### 2.3.3 Visual GUI Testing

The third generation of automated GUI based testing techniques is referred to as Visual GUI Testing (VGT) [7]. VGT is a novel script and tool based testing technique and, like its predecessors, is used to create automated GUI based tests [6]. The technique uses image recognition for GUI interactions instead of GUI component coordinates or properties like the C&R and CC&R technique. In other words, the VGT technique uses image recognition to locate and interact with GUI bitmap components, i.e. by using the actual graphics shown on a monitor [3, 6, 7, 9]. A VGT script scenario generally starts by the VGT tool giving input to the SUT, e.g. keyboard strokes or mouse clicks. The new state of the system is then observed and compared to some expected output using image recognition [7]. Because of the image recognition's ability to identify a bitmap on the GUI regardless of its position on the screen, this technique is robust to GUI layout change [3, 6, 7, 9].

VGT is supported by automated GUI testing tools like Sikuli [24], JAutomate [15] and UFT [25, 26]. In addition, these tools support the C&R technique as well [7]. Furthermore, VGT is an emerging technique in industrial practice [6, 7, 15], that treats the SUT as a black-box and is said to be non-intrusive, i.e. the technique does not require any access or even knowledge about the SUT's implementation, e.g. programming language, operating system or platform [9]. Consequently every VGT tool can interact with, and test, any application regardless if it is desktop, web or even mobile applications. Thereby, solving CC&R tools' limitation of being intrusive because they are whitebox [15].

On one hand, VGT scripts are robust to layout, API and even code changes, while on the other hand, the image recognition makes the scripts fragile to GUI graphic changes, such as change of component size, color and shape [3, 6, 7, 9]. In addition, VGT suffers from a problem that is common to all GUI based testing techniques, i.e. that the VGT scripts need to be synchronized with the SUT's state transitions [9]. This is supported by all VGT tools by adding delays to the scripts in order to synchronize the script execution with the SUT's execution. Hence, the delays are used to let the SUT's GUI reach a stable state before continuing with the test script execution [6].

Although recent studies [3] have shown that VGT is applicable for automated system testing in industry and indicated that it can be used also for acceptance testing, the technique still suffers from challenges, problems and limitations. Many of which stem from the technique's immaturity and have been shown to cause developer frustration,

add to development and maintenance costs, etc. One of the core challenges is related to the image recognition algorithms used in the tools, which is not always deterministic, leading to false-positive test results, i.e. the algorithms fail to find an image that is shown on the screen [6].

However, a study conducted by Alégroth et al. [6] shows that most of the VGT limitations discovered in their findings are perceived as manageable, which is an important result that supports the long-term industrial applicability of VGT. However, the study also states that in order to make any conclusion regarding the technique's long-term applicability, additional future research is required to fill other gaps in VGT's body of knowledge, e.g. knowledge about the technique's script flexibility, robustness and maintenance costs [6]. In addition, VGT's maintenance costs are particularly important to evaluate since it is been reported as one of the core problems for the technique's predecessors [15].

### Unified Functional Testing

The VGT tool that is used in this thesis work is Hewlett Packard's Unified Functional Testing (UFT) version 11.52 [25, 26]. UFT consists of two previous tools developed by HP, i.e. QuickTest Professional (QTP) and Service Test<sup>2</sup>. This means that UFT possesses the functionality and strengths of its predecessor QTP, but has a more user friendly GUI and is extended with additional key functionality, such as Insight Object, an image recognition based feature. The Insight Object's functionality can be used in the full range of UFT's capabilities, such as recording test cases and asserting SUT correctness. In Insight Object's recording mode, images are captured of components that are interacted with through mouse clicks and then stored in the script's object repository. In order to allow easier test maintenance, several images of the component are stored in the Insight Object automatically by the tool, during the recording mode. By using an image editor the user can select the most appropriate image of the component, as well as modify the associated interactions and the area of the picture that should be used when finding a match in the SUT. Insight Object also provides additional functionality to exclude certain areas of the component when matching the image against the SUT. This is useful for instance when the SUT's objects contain text that is non-deterministic, e.g. if the SUT supports different languages, the text part of an object like a button can be excluded to make the scripts executable for all of the SUT's supported languages. Insight Object also has a feature for determining at what similarity level a found image on the screen could be considered a match. The similarity represents the delta difference between the actual and the expected output from the SUT. This feature allows the user to define the threshold for the image recognition algorithm. This similarity level is based on a percentage that can vary between 1 and 100% similarity to the originally captured image. UFT also allows the user to manually verify what object in the application the Insight Object was matched against through a preview feature. This feature is useful

---

<sup>2</sup><http://www.starbase.co.uk/what-we-sell/hp/functional-testing/hp-service-test-software.htm>

when troubleshooting a broken script, e.g. when the tool matches an object to the wrong component in the SUT. Furthermore, this match preview highlights the object in the application with a blinking black frame around the object [25, 26].

# 3

## Context description

THE THESIS WORK WAS conducted in an industrial context at one department in CompanyX<sup>1</sup>. The company develops schedule and long term planning systems for the avionics industry. CompanyX has several departments and each department develops their own software. The department, where this thesis work was conducted, develops a software that allows airline companies to allocate and maintain their human resources, such as planning vacations, schedule trainings, and manning aircrafts. The department will be referred to as DepartmentX<sup>2</sup> in the remainder of this thesis. The system that DepartmentX develops is developed with .NET, Python and Java and the GUI contains standard .NET GUI components with additional components from a 3rd party software. The system will be referred as SystemX<sup>3</sup> for the remainder of this thesis.

The employees at Department X work according to the agile development process Scrum [27]. The department consists of two teams and within the teams everyone works both as a developer and a tester. Thus, all team members are responsible for writing tests, both unit tests and GUI-based test scripts, as well as implementing source code.

Each team has one product owner but one product manager is central for the whole department. The role of the product manager is to gather requirements from the customer while the product owners' role is to determine what features should be implemented in each sprint and to assign features to the teams. The teams share responsibility for features and every task that is put on the Scrum board is something that each team must solve together. Each sprint is 2 weeks long and the teams strive to have a new release ready for the stakeholders at the end of every week. A stakeholder is a person or an organization who affects or can be affected by the system [17], e.g. developer/tester at CompanyX and airline companies.

---

<sup>1</sup>CompanyX is kept anonymous due to a non-disclosure agreement.

<sup>2</sup>DepartmentX is kept anonymous due to a non-disclosure agreement.

<sup>3</sup>SystemX is kept anonymous due to a non-disclosure agreement.

The teams also use continuous integration and develop GUI based test scripts in parallel with the feature development. The developer who is responsible for a system feature is also responsible that one or several test script(s) are developed for it. When a test script fails, one member from the team is responsible for investigating why the test failed and either report the failure as a defect in the system or fix the test script. This team member is referred to as a test coach and team members take turns in having this role each week, e.g. a team member is only test coach for one week and then another team member takes over. This role distribution is performed in order to spread test script knowledge and distribute the burden of investigating test script failures.

At the time of the thesis work, DepartmentX used the tool QuickTest Professional (QTP) version 11.00 for their automated GUI based testing and therefore QTP version 11.00 was chosen to represent the CC&R technique in the thesis work. The department uses QTP because they get good support from HP, the tool is widely used by other companies and because there exist a lot of online documentation for the tool. However, according to CompanyX, QTP has problems, for instance with recognition of 3rd party software components and to test pop-up menus associated with right clicks. Due to the identified limitations, CompanyX is planning to switch to another HP product, namely Unified Functional Testing (UFT), version 11.52, to do their automated GUI-based testing. UFT fulfills the definition of a VGT tool, i.e. that it can interact with, and assert, a SUT through image recognition based actions [ref] and was therefore chosen to represent the VGT technique in this thesis work. Other CC&R and VGT tools were available but due to resource constraints, no other tools than QTP and UFT were included in the study. In addition, CompanyX had already purchased licenses for UFT and adding a new tools to the comparison could potentially have added additional costs for CompanyX and because of the study's time constraints adversely affected the quality of the results.

SystemX is a software system made for the avionics industry that facilitates airline companies with functionality to create and maintain long-term planning schedules for flight crews. The purpose of SystemX is to control the balance of crew supply and demand, and make more precise budgets and forecasts. The input data that the airline companies provide to the system are for example flight schedules, crew data, training, leave balances and vacations.

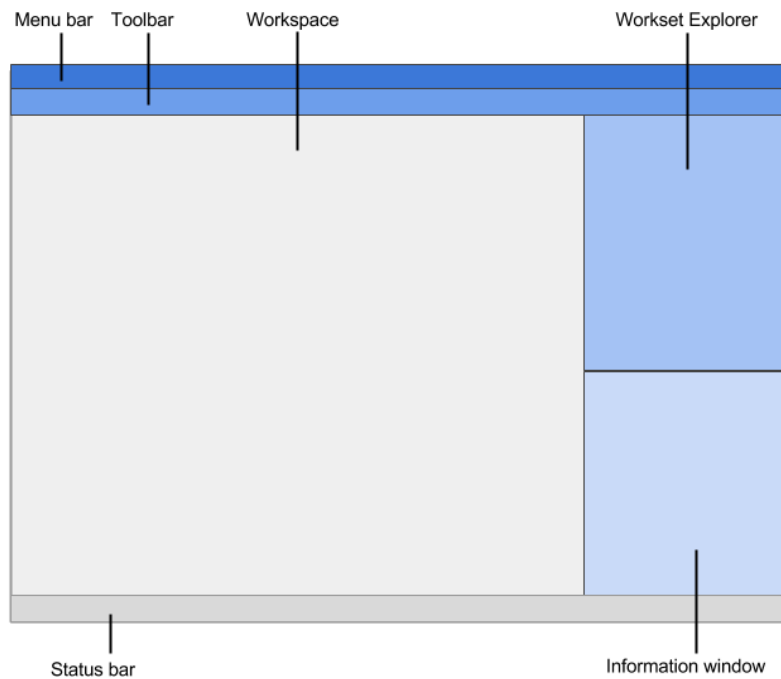
When testing SystemX, the system is executed in a test environment. This test environment always includes specific test data to be used with the latest build of SystemX. When starting up SystemX in a test environment, a window called the Launcher window is first displayed. From the Launcher window it is possible to either install the system or start it, through selections in the window. SystemX is not launched with any default data, instead the tester has to load a data workset. A workset is a subset of a complete data set provided by an airline company and is only for a specific period of time, with a defined start date and an end date. Furthermore, the system consists of 5 modules and each module contains commands and views for specific sets of data in the workset. For example, the Training module contains only the view and commands regarding the training data, i.e. data regarding training activities for the airline company, in the

workset.

The GUI client for the system is programmed with .NET while the server is based on both Python and C++. The server side of the system will not be discussed any further since it is not relevant to this thesis work because it does not have a GUI. As such, the thesis work was focused on the client of SystemX. The client's GUI consists of standard .NET GUI components with additional components from the 3rd party software DevExpress<sup>4</sup>. The GUI does not change much between releases and is not refurbished unless there is a decision to fully redesign modules. Figure 3.1 gives an overview of how the GUI is structured. On the top of the GUI is a Menu bar which contains menu items. Each menu item in the Menu bar is a button and produces a menu list when pressed. Every menu list contains several list items which represent the commands that can be performed, e.g. create, edit, open some information window, etc. The Toolbar consists of shortcut icons for actions such as save, print and open new workset. The Workspace is by default empty until the user loads a workset. When a workset has been loaded it is possible to select different views to display in the Workspace. The views can be selected from the Workset Explorer. The data in the Workspace is most often represented in grid tables or spreadsheets. With the grid tables it is possible to filter the data or do data manipulations such as create, update and delete. The GUI itself contains a lot of menus, pop-up menus and pop-up windows which appear when the user presses an icon, toolbar button or right-clicks on the grid table. Because of the many features and views of SystemX it is considered to have a rich GUI, in contrast an application like a calculator can be considered to have a lean GUI.

---

<sup>4</sup><https://www.devexpress.com>



**Figure 3.1:** An overview of SystemX's GUI.

Some of the GUI's functionalities, i.e. not well defined 3rd party components, have proved to be difficult for DepartmentX to verify with QTP. Therefore, the development and maintenance of the GUI based tests have been costly and it is desired to create a GUI-based test architecture that uses the VGT technique because it is perceived to lower development and maintenance costs, e.g. because more functionality can be tested and the perceived robustness of the UFT tool. DepartmentX has started a transition into VGT but had at the time of the thesis work only migrated a few QTP test cases into UFT test cases.



# 4

## Methodology

IN ORDER TO do an empirical comparison of the automated GUI based testing techniques CC&R and VGT, an industrial study was designed that followed the experimental guidelines of Wohlin et al. [28]. Experiments are preferable to, for instance, case studies when comparing and measuring what effect two different treatments have on a phenomenon, because they are performed in a controlled environment [28]. In this case the phenomenon was the GUI based testing and the treatments the two different GUI based testing techniques, whilst the independent variables of interest were development time, maintenance time and robustness.

However, the industrial study could not fulfill all experiment guideline criteria, and therefore the industrial study has been classified as a quasi-experiment. A quasi-experiment has the same purpose and goal as a true experiment but in a quasi-experiment the assignments of treatments, in this case the techniques CC&R and VGT, to the subjects can not be completely randomized [28]. Hence, not all influences of the dependent variables on the independent variables could be removed but were mitigated wherever possible through randomization, as explained in subsection 4.2.4.

The quasi-experiment was divided into 4 phases; Scoping, Planning, Operation and Analysis [28]. The following sections will describe the quasi-experiment process for the industrial study, with exception of the analysis phase that will be presented in section 5.

### 4.1 Scoping

The purpose of the Scoping phase was to determine and formulate the goal of the industrial study. The goal should include the following: Object of study, Purpose, Quality focus, Perspective and Context [28]. Therefore, the goal for this industrial study was defined as:

*" The goal of this industrial study was to compare the mature CC&R technique and*

*the novel VGT technique in terms of the metrics; development and maintenance cost measured as time and robustness, in order to determine which technique has the better applicability for each metric”*

## 4.2 Planning

The purpose of the Planning phase was to determine how the study would be conducted. The planning phase is divided into 7 individual steps; context selection, hypothesis formulation, variables selection, selection of subjects, experimental design, instrumentation and validity evaluation [28]. The following sections will present each step. However, the study validity has been omitted and is instead discussed in section 7.

### 4.2.1 Context selection

The context selection describes the environment in which the study, in this case quasi-experiment, will be executed [28]. The purpose of the context selection is to try to make the experiment results as generalizable as possible, i.e. to strengthen the study’s external validity. The environment can be characterized in four dimensions; offline vs. online, student vs. professional, toy vs. real problems and specific vs. general [28]. The context for the industrial study was defined as:

Study type:	Offline
Participants:	Students
Problem:	Real problems
Generalizability:	Specific

**Table 4.1:** The context definitions for the industrial study.

Firstly, the context was defined as offline since it was not planned with the purpose of customer delivery and also because CompanyX did not plan to use the test cases developed in this thesis in any real development project. Secondly, the industrial study would be conducted by students, i.e. the authors, rather than professionals. Since the study was aimed at evaluating the techniques in an industrial context, it had been preferable from a validity point of view if the study had been conducted by industrial practitioners. However, because of resource constraints it was not feasible to use industrial practitioners. Instead, the students acquired as much domain knowledge as possible such that they could perceivable represent novice engineers at the company. Thirdly, the industrial study would cover real problems rather than toy problems, since the data acquisition and technique comparison was performed based on real test cases from CompanyX for a real, yet offline, system, i.e. SystemX. Fourthly, the context was defined as specific rather than general, since the study would be performed within the industrial context of DepartmentX in CompanyX. Consequently, the results are only perceived generalizable

to similar contexts as that of DepartmentX in CompanyX and systems similar to SystemX. Therefore the context has been described in rigorous detail to ensure replicability and external validity of the results within said context.

### 4.2.2 Hypothesis formulation

Hypothesis testing is the basis for statistical analysis and the hypothesis or hypotheses need to be stated formally prior to any data collection and the study needs to be designed to ensure that the collected data is valid for answering the hypotheses [28].

For the industrial study there were three quality aspects that were of interest for the comparison between the studied tools: development time, maintenance time and robustness. However, these aspects are ambiguously defined in literature and therefore we have presented the definitions that were used during the thesis work, for each aspect, below:

*Development time (cost)* was defined as the time it took to develop and verify correct test execution per test script for one version of SystemX. The measured development time was considered more beneficial the lower it was. Hence, the most beneficial technique in terms of development time was the technique with the lower average development time.

*Maintenance time (cost)* was defined as the time it took to adjust or change a test script and verify correct test execution on another version of SystemX than the version the script was developed for. The measured maintenance time was considered more beneficial the lower it was. Hence, the most beneficial technique in terms of development time was the technique with the lower average maintenance time.

*Robustness* was defined as the chosen technique's ability (CC&R or VGT) to execute a test script despite of abnormal behavior in SystemX. The observed robustness was considered more beneficial the less action, e.g. additional script development, the tester was required to do in order to mitigate script failure due to abnormal SystemX behavior.

The hypotheses for which quantitative information were collected were therefore formulated to allow for formal statistical analysis, e.g. with a T-test or Wilcoxon rank sum test. The reason for the choice of said type of tests was because it was not known prior to the thesis work which technique would be the least costly or most robust. Thus motivating a need for unbiased hypotheses, as given by the above stated tests, to gain an unbiased view of which technique was more effective for each quality aspect and as a whole based on all collected results. The following hypotheses were defined for the industrial study:

#### Development time

- $H_{01}$  : *There is no difference in average development time between the CC&R technique and the VGT technique.*

- $H_{11}$  : *There is a difference in average development time between the CC&R technique and the VGT technique.*

$$H_{01} : \mu_{CC\&R} = \mu_{VGT}$$

$$H_{11} : \mu_{CC\&R} \neq \mu_{VGT}$$

#### Maintenance time

- $H_{02}$  : *There is no difference in average maintenance time between the CC&R technique and the VGT technique.*
- $H_{12}$  : *There is a difference in average maintenance time between the CC&R technique and the VGT technique.*

$$H_{02} : \mu_{CC\&R} = \mu_{VGT}$$

$$H_{12} : \mu_{CC\&R} \neq \mu_{VGT}$$

#### Robustness

- $H_{03}$  : *There is no difference in average robustness between the CC&R technique and the VGT technique.*
- $H_{13}$  : *There is a difference in average robustness between the CC&R technique and the VGT technique.*

$$H_{03} : \mu_{CC\&R} = \mu_{VGT}$$

$$H_{13} : \mu_{CC\&R} \neq \mu_{VGT}$$

### 4.2.3 Variables selection

Before creating the experimental design for the industrial study it was necessary to choose the dependent and independent variables. The independent variables are the variables that can be controlled and changed in the experiment and dependent variables are variables that are affected by manipulating or changing the independent variables [28]. The dependent variables are the variables that will be measured and were therefore categorized as:

#### Independent variables

- CC&R
- VGT

#### Dependent variables

- Development time
- Maintenance time
- Robustness

#### 4.2.4 Selection of subjects

In the Context selection, section 4.2.1, it was stated that the industrial study would be conducted by the authors themselves, due to resource constraints, which lead to convenient sampling of the study's subjects. This sampling is also the reason why the industrial study was classified as a quasi-experiment instead of a true experiment, since the selection of subjects could not be randomized as prescribed for a true experiment.

However, the test case selection and implementation was randomized. CompanyX provided the authors with natural language design specifications for 40 valid test cases for SystemX that had previously been implemented in QTP. The test cases' design specifications included pre-conditions and several test steps for each test. The pre-conditions contained criteria, i.e. the state that SystemX must fulfill before executing the test steps. For example, a typical pre-condition was that SystemX was started and loaded with a certain workset. Furthermore, each test step described an action that the tester should perform on SystemX and the expected result of that action. An example of what a design specification could look like is presented in Table 4.2.

<b>Precondition 1</b>	SystemX started	
<b>Precondition 2</b>	Workset X loaded	
<b>Test Steps</b>	<b>Action</b>	<b>Expected results</b>
<b>Step 1</b>	"Click on the Crew icon"	"Crew view is displayed"
<b>Step 2</b>	"Select crew xyz"	"Crew xyz is selected"
...	...	...
<b>Step N</b>	...	...

**Table 4.2:** Example of a test case design specification was structured.

After the design specifications had been obtained, the test cases were categorized into four categories; long, short, simple and complex. A test case was categorized as long if it contained five or more test steps, because the average test steps of the provided 40 design specification was 4.8. Furthermore, a test case was categorized as complex if any test step contained data manipulation, e.g. create, update and delete, or an assertions of the correctness of a table value. Data manipulation and table assertions were categorized as complex since they were perceived to be cumbersome to implement with both techniques prior to the thesis work. Each test case got two categorization and belonged to one of the following groups; long and simple, long and complex, short and simple and short and complex

After all 40 test cases had been categorized according to this scheme, 20 test cases were selected at random to be implemented in the two techniques. In addition, analysis of the set of 20 implemented tests showed that the minimum number of test steps in a specification was two, the maximum number was 11 and the average number was 5.5, which is perceived to cover the range of most general test cases for SystemX. The reason

why not more than 20 test cases were chosen for implementation was due to resource constraints.

The distribution of categorizations was seven Short and Simple, three Short and Complex, four Long and Simple and lastly six Long and Complex test cases. The resulting sample was perceived suitable for the thesis work due to its even distribution among the four categories.

<b>Categorization</b>	<b>No. of test cases</b>
Long & Complex	6
Long & Simple	4
Short & Complex	3
Short & Simple	7

**Table 4.3:** The categorization of the 20 test cases.

#### 4.2.5 Experimental design

According to Wohlin et al. [28] there are three general design principles for an experiment or a quasi-experiment and most of them use some combination of these principles; randomization, blocking and balancing. The following section will explain what design principles were applied in the industrial study.

##### **Randomization**

In an experiment or a quasi-experiment randomization can be used to select subjects to represent the population of interest and to distribute objects among the subjects [28]. As mentioned before, the industrial study was classified as a quasi-experiment, since there were only two human, uncontrollable, subjects and the objects, i.e. test cases, which were not randomly assigned to the subjects. Rather, a plan was created in order to divide the usage of the techniques between the subjects. The plan consisted of switching the testing tools between subjects after a test case had been implemented, as can be seen in Table 4.4. The plan was expected to reduce the learning effect of a subject only using one tool consistently, which could have affected the results by consistently lowering the development time for one of the techniques/tools. However, the drawback of the test implementation plan was that it could take time for the subjects change mindset after switching from one technique to the other, which is perceived to have had an affect on the measured development time. However, since the used tools have many similarities, this switch effect is considered negligible.

Test case	Subject A (Author A)	Subject B (Author B)
1	QTP (CC&R)	UFT (VGT)
2	UFT (VGT)	QTP (CC&R)
3	QTP (CC&R)	UFT (VGT)
4	UFT (VGT)	QTP (CC&R)
5...n	Switch tool	Switch tool

**Table 4.4:** Visual representation of test case development procedure used.

### Balancing

Quasi-experiments and experiments have a balanced design when the treatments in the experiment have equal number of subjects [28]. By having a balanced design the statistical analysis of the data becomes more simple and the analysis is perceived to be stronger than an analysis from a non-balanced design [28]. This industrial study had a balanced design, due to the subjects developed the same amount of test cases for each technique.

### 4.2.6 Instrumentation

There are three types of instrumentation for a quasi-experiment or an experiment; instrumentation objects, guidelines and measurement instruments [28].

For the industrial study, guidelines were created for the data collection. The guidelines described the data collection process that was used during script development, i.e. what data to collect and when to collect it. In addition the guidelines specified when the development of a test from the test design specification was considered completed.

A spreadsheet was used as the measurement instrument for the quantitative data collection for the techniques' script development time, maintenance time and robustness measurements. The data collection guidelines described in the previous paragraph helped ensure that the data collection was performed systematically. For definitions of collected metrics the reader is referred to the Hypothesis formulation in section 4.2.2.

The script development and execution was performed on laptop computers with 64-bit Windows 7 operating systems, each with 4 GB RAM and Intel i7-2620 CPUs clocked at 2.70 Ghz. Furthermore, the screen resolution that was used during script development was 1600x1024 pixels. The resolution of the computers is added here since it is perceived to affect the speed of the image recognition as well as the size of the captured images in UFT.

## 4.3 Operation

During the Operation phase, the industrial study was executed and the comparative data about the two respective techniques CC&R and VGT was collected. The Operation

phase is divided into 3 individual steps; Preparation, Execution and Data validation. The following subsections describe each of the steps in the industrial study [28].

### 4.3.1 Preparation

The purpose of the preparation step is to select the participants that will act as subjects and prepare materials needed for the execution step [28]. As previously stated, the industrial study would be performed by the authors of this thesis, due to resource constraints. However, the authors needed training in QTP and UFT, since they had limited hands-on experience with the tools. Therefore, an analysis of the tools was conducted and it consisted of inspection of the tools' development environment, review of the tools' documentation and following tutorials. Overall time spent on working with the tools before the industrial study was about 20 hours per subject in each tool. Furthermore, the subjects attended an educational workshop, held by CompanyX, about SystemX to learn about the system's features from an end-user perspective. The authors' goal of attending the workshop was to gain domain knowledge about SystemX that would be useful to understand the manual test specifications.

DepartmentX releases a new version of SystemX every two weeks and since the study would stretch over several releases in calendar time an offline version of the system was created that would not be affected by the daily builds or the releases, i.e. a fixed version of the system. Thus ensuring that the functionality of the system would not change during the development of the test cases. The fixed version of SystemX will in the continuation of the thesis be referred to as the first version of SystemX.

However, in order to collect information about the maintenance time associated with the CC&R and VGT test cases, another version of SystemX was needed. Therefore, a version from three releases after the first version was used when measuring maintenance time. The later version of SystemX will be referred to as the second version of SystemX in the remainder of the thesis.

### 4.3.2 Execution

The industrial study was conducted over eight weeks, i.e. four releases of SystemX. During these weeks the test scripts were developed and maintained and measurements were collected for the metrics development time, maintenance time and robustness. The following section will describe the data collection process for each metric.

#### Development

The development time metric was defined as the time it took to develop a script for one test case, based on the manual test design specification and verify that it executed correctly. The test script implementation was defined as complete when all test steps in the design specification had been implemented and the entire test case executed successfully against the first version of SystemX. However, the measured development time only includes the time when the subject was actively developing or executing a test case. As such, development time was not recorded if the subject left his workstation.



Every developed test script began by launching SystemX and ended by terminating it. This test script design was chosen in order to make the test cases consistent but also independent of one another. Thus, allowing the test cases to be executed sequentially or in random order.

As described in the Context description, section 3, it was identified that the testers at DepartmentX did not implement functionality directly into Actions in QTP. Instead they created a function library which stored the functions used in the test scripts. This practice was performed to promote re-usability of recurring actions since many test cases can perform the same actions, e.g. launch the system, open certain menus, etc. In order to further capture the context of CompanyX, the company's script development practice with actions stored in functions was also used during the study. Each function consisted of an action to be performed on SystemX and an assertion of the expected results of said action. For example, opening a certain view in SystemX's GUI and verifying that the view was successfully opened.

The functions were stored in a function library and could therefore be reused among test cases. When a function was created for a test case, the development time for that function was documented. Thereafter, when the function was reused in another test case, the development time for that function was added to the total development time of the currently developed test case. Reusing functions without adding the function's development time would have otherwise resulted in unequal development time among the test case where the function was developed and later where it was reused.

### **Maintenance**

When all 20 test cases had been implemented in both QTP and UFT, they were executed on the second version of SystemX. All failing test cases were then analyzed and test cases that could be maintained were fixed. Maintenance time was measured as the time it took to adjust or re-write a test case until it could execute successfully on the second version of SystemX.

However, in the second version of SystemX one module of the system had been completely redesigned, new functionality had been added and some old functionality had been removed. These changes had adverse affects on the applicability of nine of the developed test cases that were no longer applicable because the functionality they aimed to test had been removed from the system. The changes to the system therefore affected the number of data points that could be collected for both of the technique's maintenance costs, i.e. only 11 data points out of 20 developed scripts. Consequently, the lack of data points presents a threat to the validity of the results and therefore the reader is informed every time the results refer to the maintenance data set. This outcome was unforeseeable prior to the start of the study, but was recognized as an unlikely general threat, i.e. worst case, since the study was performed at a company with its own, and to the authors of this thesis, unknown agenda.

### **Robustness**

Robustness was defined as the technique's ability to successfully execute a test case de-

spite of unexpected or abnormal behavior in SystemX. The data collection for robustness was conducted by executing the 20 developed test cases sequentially 10 times in a row for each technique. The sequential execution was performed by creating test suites in respective tools where the core of the test suites consisted of a method that loaded and executed each test case from a given directory.

When a failure occurred during the execution of a test suite, the cause of the failure was investigated manually in order to identify if the failure was caused by unexpected behavior of SystemX or a defect in the system. If the former was the cause of the failure, the failure was documented and the test suite execution was resumed from the next test case in the suite. Otherwise, the defect was reported to DepartmentX. The failure analysis was important in order to identify the number of false positives and negatives produced by the techniques and in extension the techniques' individual robustness.

### 4.3.3 Data validation

According to Wohlin et al. [28], after the data has been collected in an experiment or quasi-experiment, the researcher must check that the data is reasonable and that it has been correctly collected. In the industrial study, both the subjects used the same systematic approach to collect data, which also included data validation. Furthermore, two test engineers at DepartmentX agreed to verify the correctness of the developed test cases. The test engineers' verification consisted of executing the test cases in both tools on SystemX and checking that the test cases' assertions were correct according to their design specification.

# 5

## Results

**T**HE FOLLOWING SECTION presents the results of the thesis work which were obtained during the industrial study.

### 5.1 Development

As previously stated, development time was measured as the time it took to successfully implement a test case for the first version of SystemX. A test case was defined as implemented when all the test steps of the test design specification had been implemented and the test terminated successfully after completing all its steps. Table 5.1 presents the development time of each test case, in minutes, and also their respective categorization, see subsection Selection of subjects in the Planning section 4.2.4. The tests are presented in the chronological order they were implemented in the study.

Test case	Category	CC&R development time (minutes)	VGT development time (minutes)
1	Short, Simple	76	87
2	Short, Complex	155	154
3	Long, Simple	99	98
4	Long, Complex	104	75
Continued on next page			

Table 5.1 – continued from previous page

Test case	Category	CC&R development time (minutes)	VGT development time (minutes)
5	Long, Complex	138	106
6	Short, Simple	105	65
7	Long, Complex	163	122
8	Long, Simple	155	146
9	Short, Simple	113	90
10	Short, Simple	30	32
11	Long, Complex	139	106
12	Short, Simple	44	43
13	Long, Complex	121	104
14	Short, Simple	121	72
15	Short, Complex	66	52
16	Long, Simple	117	60
17	Short, Simple	100	74
18	Short, Complex	107	58
19	Long, Complex	87	101
20	Long, Simple	133	90
<b>Total</b>		2174	1734
<b>Average</b>		109	87
<b>Standard deviation</b>		36	32

**Table 5.1:** Table showing development time in minutes, for all test cases in each technique respectively, for the first version of SystemX. The table also shows the categorization of all test cases.

In order to provide the reader with a descriptive overview of the measured development times the data from Table 5.1 has been visualized in Figure 5.1. The dark blue line in Figure 5.1 represents the development times for VGT and the light blue the development

times for CC&R. For most of the test cases, the two lines have synchronized peaks and troughs, i.e. if the development time is high for a test case developed in one of the techniques, the development time is also high for said test case in the other technique. Furthermore, some test cases have notably similar development times in both techniques. Attempts were made to find some commonality between the test cases by inspecting their categorizations and through analysis of their designs steps to explain the curves. However, no general conclusion could be made.



**Figure 5.1:** Graph showing a visual representation of the development time for all test cases in each technique respectively, for the first version of the system.

Figure 5.2 presents the accumulated development time, i.e. the total time spent on developing the test cases in each technique for the first version of SystemX as new test cases were incrementally added to the techniques' test suites. As shown in Figure 5.2, the accumulated development time for CC&R is higher than for VGT and the difference between the lines increases steadily throughout the graph. This can be explained by the CC&R technique's average test case development time of 109 minutes compared to the VGT technique's 87 minutes per test case. Thus, indicating that the development costs of CC&R scripts are higher than for VGT scripts in contexts similar to CompanyX and SystemX.



**Figure 5.2:** Graph showing a visual representation of the accumulated test development time trends for each technique, for the first version of SystemX. The test development costs have been plotted in the chronological order they were implemented.

### Hypothesis testing

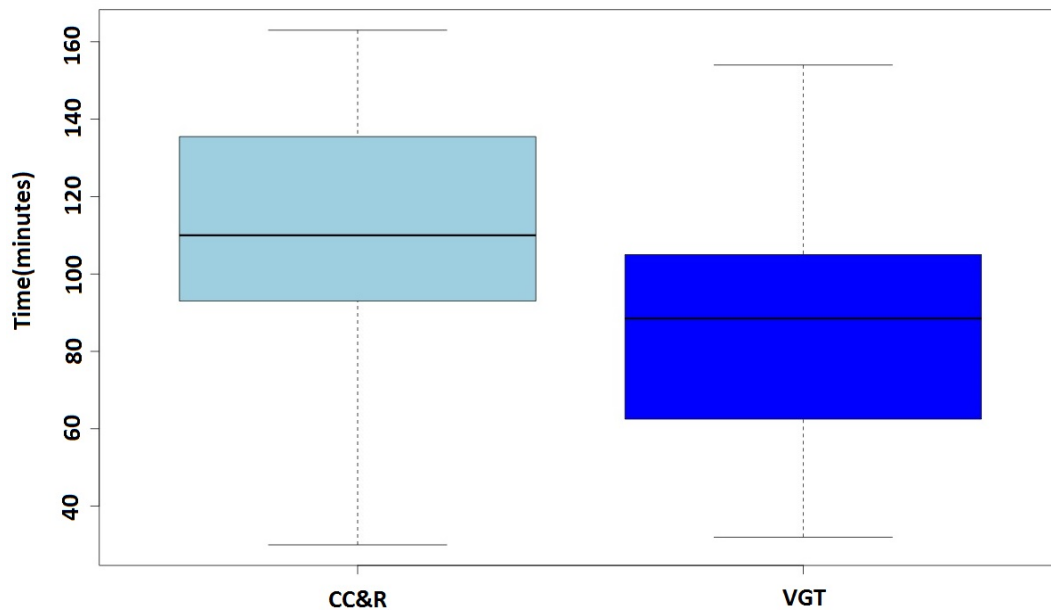
The null hypothesis for comparing average development time between the two techniques, CC&R and VGT, was as follows:

$H_{01}$  : *There is no difference in average development time between the CC&R technique and the VGT technique.*

$$H_{01} : \mu_{CC\&R} = \mu_{VGT}$$

In order to statistically verify this hypothesis a Shapiro-Wilks test was first conducted to determine if the data was normally distributed or not. The test resulted in a p-value of 0.5469 for the CC&R data sample and a p-value of 0.812 for the VGT data sample. These results are over the 0.05 threshold and therefore show that the data is normally distributed in both samples [29]. A conclusion that is further supported by the box plot in Figure 5.3 that shows that the data points from both samples are centered around the median, which also suggests that the data is normally distributed. The box plot also shows the dispersion and skewness of the collected samples. The line in the middle of the box plot indicates where median of the sample is located and the box's top and bottom lines represent the 75% quantile and 25% quantile respectively. The tails of the box represents the theoretical bound within which it is most likely to find all data points if the distribution is normal [28]. Moreover, the box plot shows that both samples have no outliers and that the majority of both sample's data points are concentrated around the median. Furthermore, the fact that both samples have no outliers indicates that the chosen test cases are representative for all the 40 manual test cases that were available

for SystemX. Because the data sets were found to be normally distributed, the null hypothesis,  $H_{01}$ , can be tested with the parametric Student T-test rather than the non-parametric Wilcoxon ranked sum test. When comparing two groups, the T-test assumes that; both groups' samples are random, independent, have unknown but equal variances and come from a normally distributed population [29]. Analysis of our data samples showed that they fulfilled the assumptions of being random and independent. However, in order to determine if the two groups had equal variance, an F-test was conducted. The F-test resulted in a p-value of 0.6445, which let's us accept the F-test's null hypothesis that both data sets have equal variance. The T-test was performed with a 95% confidence interval and had a p-value of 0.04706. Although this value is very close to 0.05 we are still forced to reject the null hypothesis  $H_{01}$  and conclude that there is a statistically significant difference in average development time between the two techniques [29]. As such, the probability of committing a Type I error, i.e. incorrect rejection of a true null hypothesis, is 5%. However, the T-test had a power of 56,6% which states the probability of correctly rejecting the null hypothesis when the null hypothesis is false. Moreover, the effect size, which describes the magnitude of difference between the two groups, was calculated to be 0,648. The effect size means that difference between VGT and CC&R is of a magnitude of 64,8% of a standard deviation of about 36 minutes [29].



**Figure 5.3:** Box plot visualizing the dispersion and skewness of the development time samples, for each technique.

## 5.2 Maintenance

After all 20 test cases had been developed and successfully verified for the first version of SystemX, they were executed on a second version of SystemX. As previously mentioned, the second version of SystemX was an updated release of the first version. Maintenance time was then measured as the time it took to adjust or re-write the tests cases that failed on the second version of SystemX until they executed successfully on the second version of SystemX. Table 5.2 shows the results of the maintenance time measurements in minutes as well as the total maintenance time for each technique. The test cases that were not applicable for the maintenance measurements, i.e. could not be maintained for the second version of SystemX due to changes to the system, are marked with N/A. The total maintenance time for the two techniques is remarkably low, only 18 minutes for the CC&R technique and 36 minutes for the VGT technique to be compared to the total development time of 2174 minutes for CC&R and 1734 minutes for VGT . However, the table also shows that there were 9 out of 20 test cases that were not applicable (N/A) for the second version of SystemX, which made it impossible to collect any maintenance measurements for said test cases. In addition, the lack of maintenance measurements of the N/A tests also explains the low overall maintenance time of the two test suites.

Test case	CC&R maintenance time (minutes)	VGT maintenance time (minutes)
1	7	20
2	11	13
3	0	0
4	N/A	N/A
5	N/A	N/A
6	N/A	N/A
7	0	0
8	0	0
9	N/A	N/A
10	0	0
11	0	0
12	0	0
Continued on next page		



**Table 5.2 – continued from previous page**

Test case	CC&R maintenance time (minutes)	VGT maintenance time (minutes)
13	N/A	N/A
14	N/A	N/A
15	0	2
16	N/A	N/A
17	N/A	N/A
18	N/A	N/A
19	0	0
20	0	0
<b>Total</b>	18	35
<b>Average</b>	1.5	3.5
<b>Standard deviation</b>	3.67	6.89

**Table 5.2:** Table showing the maintenance measurements for all tests, in both techniques. N/A means that maintenance of the test case was not applicable, due to major functionality changes between the two version of the system.

### Hypothesis testing

The null hypothesis for comparing the average maintenance time between the two techniques, CC&R and VGT, was defined as:

$H_{02}$  : *There is no difference in average maintenance time between the VGT technique and the CC&R technique.*

$$H_{02} : \mu_{CC\&R} = \mu_{VGT}$$

As for the development time measurements, a Shapiro-Wilks test was performed to determine if the data was normally distributed. Note that the test cases that were not-applicable for the second version of SystemX were excluded from the data set during the test. Thus, resulting in a sample size of only 11 maintenance time measurements. The Shapiro-Wilks test resulted in a p-value of 2.102e-06 for the CC&R data sample

and  $6.482e-06$  for the VGT data sample, which is lower than the 0.05 threshold [29]. Therefore we must reject the null hypothesis and conclude that the data is not normally distributed.

Because the data was not normally distributed we instead used the non-parametric Wilcoxon rank sum test to test the null hypothesis,  $H_{02}$ . When comparing two groups, the Wilcoxon rank sum test assumes that the data comes from the same population, is independent and not paired. However, the test does not assume that the data comes from a normally distributed population but that it is at least ordinal [29]. Analysis of our data samples showed that they fulfilled the assumptions of coming from the same population, being ordinal, independent and not paired.

The Wilcoxon rank sum test was performed with a 95% confidence interval and had a p-value of 0.5916. Since this p-value is greater than the 0.05 threshold, it is not possible to reject the  $H_{02}$  hypotheses. Moreover, the test had a power of 14.5%, which in turn means that the probability of committing a Type II error is as high as 85.5%. When a Type II error occurs, we erroneously fail to reject the null hypothesis, although the null hypothesis is false. In our case this means that the probability of erroneously failing to recognize a difference in average maintenance time between the techniques is 85.5%. However, the low power is an indirect result of our small sample size that originally was 20 but ended up being only 11, as previously mentioned. Therefore, any statistically claim regarding the average maintenance time is still open to speculation. Nevertheless, based on the statistical results for the null hypothesis  $H_{02}$  we conclude that there is no statistically significant difference in average maintenance time between the two techniques.

### 5.3 Robustness

In order to obtain the robustness measurements for the techniques' scripts, the 20 test cases were grouped into a test suite that was executed 10 times on the first version of SystemX. Afterwards, the average number of successful test case executions for each technique was calculated out of the total 200 runs, results shown in Table 5.3. However, because some robustness problems had been observed during test development, each test case was also executed 10 times individually. Table 5.3 only presents the robustness measurements when running the test cases sequentially in a test suite, as the individually executed test cases' robustness measurements resulted in 10 successful runs out of 10 for all test cases in both techniques.

The reason why the measurements in Table 5.3 represent robustness is because we expect the test cases to be successfully executed 10 out of 10 times. Hence, the results presented in Table 5.3 are indicative of robustness issues with the techniques' different approaches. Thus, the results indicate that VGT is less robust than CC&R and that image recognition is more prone to failure than object property identification. The source of the robustness problem is purely technical, i.e. image recognition is complex and the technique is immature. Based on this observation we stipulate that as the technique matures and image recognition algorithms are improved, the robustness of VGT scripts

should improve as well. Support against this statement can however be derived from the robustness measurements for the more mature CC&R technique that had less than a 100 percent success rate. It is perceived that the technique's robustness issues, as for VGT, can be solved through technical advances but also allows us to speculate that it may be a long time, if ever, before VGT's robustness is perfected. A statement that is derived on the similarities of the two approaches. However, the results show that further research is required in order to improve the robustness of both techniques.

During the execution of the test suites, unexpected failures occasionally occurred. The CC&R test suite had one test case that failed unexpectedly twice during the test suite execution. The failure was because of object mapping failure of a component from the 3rd party software, DevExpress, that was not well defined in the object repository.

For the VGT test suite there were two types of unexpected failures that were re-occurring. Firstly, test execution would fail when a VGT test case performed an action to open a pop-up window, because SystemX's launcher window would be placed in focus and cover it.

Secondly, the test execution would fail when a VGT test case performed an action to interact with, e.g. click, a GUI component in a pop-up window but instead only managed to place the pop-up window in focus without any interaction being performed. Both failures with the VGT test suite occurred sporadically for different pop-up windows and test case executions.

The failures in the VGT test suite could be fixed by adding failure mitigation code in the test cases' implementation. However, these test cases did execute successfully when they had been developed. Therefore, it was expected that the test cases would execute successfully every time they are executed, even though they are being executed in a test suite. Thus, the additional failure mitigation was not perceived as necessary for the robustness measurements.

<b>Test case</b>	<b>CC&amp;R successful executions</b>	<b>VGT successful executions</b>
1	10	10
2	10	8
3	10	10
4	10	10
5	10	10
6	10	9
7	10	9
8	10	9
Continued on next page		

Table 5.3 – continued from previous page

Test case	CC&R successful executions	VGT successful executions
9	10	9
10	10	10
11	10	9
12	10	10
13	10	8
14	10	10
15	10	8
16	10	10
17	10	10
18	10	10
19	10	9
20	8	9
<b>Total</b>	198	187
<b>Average</b>	9.9	9.35
<b>Standard deviation</b>	0.48	0.75

**Table 5.3:** Table showing the robustness measurements for all test cases, for both techniques, when executing all test cases sequentially in a test suite. The test suite was executed 10 times for both techniques.

### Hypothesis testing

The null hypothesis for comparing the average robustness of the techniques was defined as:

$H_{03}$  : *There is no difference in average robustness between the CC&R technique and the VGT technique.*

$$H_{03} : \mu_{CC\&R} = \mu_{VGT}$$

As for the previous hypotheses tests we first conducted a Shapiro-Wilks test to determine if the data was normally distributed. The tests resulted in a p-value of 2.693e-09 for the CC&R data sample and 0.000273 for the VGT data sample. These results are lower than the 0.05 threshold and therefore show that the data is not normally distributed [29].

Because the data was not normally distributed we once again picked the non-parametric Wilcoxon rank sum test in favor of the Student T-test to test the null hypothesis  $H_{03}$ . As previously stated, the Wilcoxon rank sum test assumes that the data comes from the same population, is independent and not paired. However, the test does not assume that the data comes from a normally distributed population but that it is at least ordinal [29]. Analysis of our data samples showed that they fulfilled the assumptions of coming from the same population, being independent and not paired.

The Wilcoxon rank sum test was performed with a 95% confidence interval and had a resulting p-value of 0.002965. Because of this result we must reject the null hypothesis  $H_{03}$  and conclude that there is statistical significant difference in average robustness between the two techniques. Furthermore, the test had a power of 80,8% and an effect size of 0.89502722. This means that there exist statistical difference between the average robustness of CC&R and VGT scripts in successful executions with a magnitude of 89,5% of a standard deviation of about 0,5 test cases [29]. Thus, supporting that our conclusion regarding  $H_{03}$  is correct.

# 6

## Discussion

The following section presents the discussion regarding the acquired results from this thesis work. More specifically the section will discuss each measurement result, i.e. development time, maintenance time and robustness, as well as a more general discussion regarding the implications of this work. In addition, a discussion regarding the used testing tools will also be presented which is based on the individual experiences and observations made by the authors during the execution of the industrial study.

### 6.1 Development

The development time results, presented in subsection 5.1, show that there is a statistically significant difference between the two techniques' average development time. A conclusion that is supported by the VGT technique's total development time being 440 minutes less than the CC&R technique's total time. Further support is given by CC&R's average development time of 108.70 minutes compared to VGT's 86.69 minutes. Quantitative results that indicate that the average total development time of VGT scripts is not only statistically significantly different but also lower than the development time of CC&R scripts.

Furthermore, we argue that there exists also a difference in what types of tests that are advantageous to develop in each technique, i.e. advantageous in terms of obtaining the lowest development time possible. In addition, we argue that how scripts are developed in each technique, i.e. the development process, affects the development time as well, a statement that is supported by previous work [6].

When implementing the test scripts using the VGT technique with the tool UFT, a repetitive development pattern appeared. Firstly, we visually localized the component to be interacted with, in SystemX's GUI. Secondly, we created a snapshot of that component by using UFT and saved it as an Insight Object in the tool's repository. Thirdly, we used that object in the test script, in association with an action such as

an assertion that the object existed on SystemX's GUI by using image recognition or invoking a supported method, such as a click or text input. This recurring work process enabled us to follow the test design specification and intuitively implement and verify the specified test steps in a consistent manner. Furthermore, because of the VGT technique's properties, intuitiveness, and repetitive work process, the test developer does not require any specific system knowledge nor needs to spend time on planning the test case creation. In contrast, the CC&R technique's work process is more dynamic and alternating, which requires the test developer to both make use of system knowledge to implement a test case as specified in the test design specification. For example, let's assume that a design step in a test design specification specifies that the columns in a certain table should have some specific names and follow a certain order. With the VGT technique the process would still be carried out as previously described, i.e. we would save a snapshot of the table and its columns in the SUT and create an assertion to verify that the snapshot exists in the SUT. In contrast, with the CC&R technique we first have to figure out how to access the column names and then how to verify them accordingly. This process is perceived as a much more technically daunting task than the intuitive VGT approach of using the SUT's graphical output. Determining how to access component information is as such neither straightforward or even possible in some cases. For instance, it is not possible to acquire component properties when the SUT is written in several programming languages or if the SUT is distributed over several computers, etc. As such, CC&R is dependent on the SUT's implementation which also explains why system knowledge is required to write scripts in the CC&R technique.

Furthermore, the planning process for the CC&R technique takes a varying amount of time, i.e. the time it takes depends on the type of assertions that the test design specification includes, how the SUT is implemented and the test developer's experience and skills. However, if the CC&R implementation is planned correctly and made as general as possible, the implemented solutions can be reused, which lowers the total development time of a CC&R test suite. To exemplify, in order to perform an assertion of a column title in a data table, the test developer could create a function that is able to verify any given table's column titles, taking the table and the sought text as input. However, if there is no benefit in creating a general solution to the test, i.e. if the assertion is only performed once in the entire test suite, the cumbersome process of planning and implementing the general test case in the CC&R technique would still remain the same, in contrast to the straight forward static process previously presented for the VGT technique.

Thus, we suggest that the techniques are respectively advantageous for different types of tests and that the work processes coupled with each technique affect the development time. However, we also recognize that further research is required to evaluate what types of tests are better served by one technique over the other. We also recommend that script development time should be a key metric in this future work.

## 6.2 Maintenance

The results from the maintenance measurements in subsection 5.2, show that there is no statistical difference in average maintenance time between CC&R and VGT. However, the conducted Wilcoxon rank sum test had only sample sizes of 11 and a statistical power of 14.5%. The reason for these low sample sizes was because several of the developed test cases were no longer applicable because of a redesign of a key module in the second version of SystemX.

Previous studies claim that between 30 to 70% of CC&R test scripts require maintenance even after minor changes to the SUT [11, 12, 18]. As such we expected the maintenance time to be high for the CC&R technique. However, our results show that the CC&R technique required only a total of 18 minutes of maintenance, compared to the CC&R total development time of 2174 minutes, i.e. the maintenance was only 1.57% of the total development time. In turn, the VGT technique required 36 minutes of maintenance, i.e. 3.5% of the total development time of the VGT scripts. Consequently, these results show that both of these techniques are associated with low maintenance time compared to development time, even though it was only for 11 test cases.

However, since only 11 measurement points were acquired for the maintenance evaluation we suggest that further research is needed to support our results regarding the differences in maintenance cost between the two techniques. Furthermore, we suggest that future research into the maintenance cost of the studied techniques should be conducted over a longer period of time and also evaluate the maintenance costs associated with newly implemented features.

## 6.3 Robustness

The results in section 5.3, show that there exist a statistical significant difference in robustness between the CC&R technique and the VGT technique. The results of the test suites' executions showed that the CC&R technique only had two failed test case executions while the VGT technique had 13.

However, it is possible to argue that these failures are not caused by the techniques but rather the tools. The CC&R test suite failure was caused by object mapping failure. This failure occurred when QTP failed to match a component from the 3rd party software, DevExpress, with the object in the repository due to an insufficient object description.

In VGT, there were two test suite failures that were re-occurring. First, the Launcher window appeared in front of a pop-up window as it was being opened during a script execution, and second, some interactions with pop-up windows were not executed correctly. We argue that these script failures were caused by immaturity in the image recognition functionality of UFT. Similar problems were identified by Alégroth et al. [6] when execute a VGT test suite with the open source VGT tool Sikuli [24]. Therefore, it can be argued that current tools are not suitable to handle the execution of a VGT test suite and need to be further improved if the VGT technique is to replace or complement



current testing techniques.

Furthermore, it is important to recognize the purpose of the test cases and discuss whether they were created with the intention to verify or validate the system, i.e. used for system testing or acceptance testing. Verification of a system consists of checking if the system conforms with the system's requirements specification whilst validation seeks conformance to the end customer's needs [30]. Assuming that the test cases implemented during this study had been created to validate system conformance, the reported VGT test suite failures of SystemX would be acceptable, since these behaviors would not be accepted by a human user either. In contrast, if the test cases were supposed to verify the system according to the system's specifications, then the VGT test suite failures would not be acceptable, since it would be desired to run the system regardless of any unexpected interference of pop-up windows or pop-up messages.

Consequently, we argue that there exist a trade-off between creating GUI based tests to verify a system or to validate it, i.e. a system test through the GUI or an acceptance test. If a GUI based test is created with the intention to verify a system, then an extra failure mitigation needs to be added manually in a VGT script, in order to handle the unexpected behaviors of different windows and pop-up windows. However, this failure mitigation would only be ad-hoc, e.g. by adding an extra assertion or an extra click. In contrast, this failure mitigation would not be needed for a CC&R script, because the script inherits robustness to interact with components and windows from how the technique works. The technique can interact with the components and windows even though they are not visible on the screen, due to the technique can access components' properties and methods. Therefore, we agree with Börjesson and Feldt [3] that there is a need for a framework or guidelines in order to develop more robust VGT test scripts and to achieve the same level of robustness as for the CC&R technique's scripts.

## 6.4 General

The sub-goal of this thesis was to determine if it is beneficial for CompanyX to change their current GUI based testing technique from CC&R to VGT. Based on our findings, trade-offs between development time, maintenance time and robustness needs to be considered. Table 6.1 presents a summary of which technique we consider to be more beneficial in terms of the investigated quality aspects development time, maintenance time and robustness.

As can be seen in Table 6.1, we believe that it is more beneficial to use the CC&R technique when prioritizing maintenance time and robustness as the CC&R scripts had lower total maintenance time and higher robustness than the VGT scripts. In contrast, as can be seen in Table 6.1 we believe that it is more beneficial to use VGT when prioritizing development time, as VGT had lower total development time than CC&R. In addition, SystemX contains several 3rd party components that are not well defined and require image recognition to verify, which the CC&R technique does not support.

Therefore we argue that a trade-off between the investigated quality aspects must be made when selecting a technique. Nevertheless, because the UFT tool supports both the

CC&R and VGT technique we suggest CompanyX to use UFT instead of QTP. Moreover, we suggest CompanyX to use the most suitable technique for different types of test cases but mainly continue using their current GUI based testing technique which is CC&R as it will be most beneficial for the most part of their testing of SystemX. Furthermore, at CompanyX the test engineers are not the only ones responsible for creating automated GUI based test scripts, the developers do so as well. This is an additional reason why it is more beneficial for CompanyX to use CC&R as the technique's work process allows the developers to advantageously make use of their coding skills to create general and reusable test architecture, which we have previously discussed in section 4.3.2.

Technique	Development time	Maintenance time	Robustness
CC&R		X	X
VGT	X		

**Table 6.1:** The evaluation of which technique is considered to be more beneficial in terms of the investigated attributes development time, maintenance time and robustness or GUI based testing of SystemX.

No previous research has conducted a comparison of the CC&R and the VGT technique in an industrial context, to the authors' best knowledge. Hence, our presented findings contribute to the academic body of knowledge in the field of automated GUI based testing by showing the benefits and drawbacks of the two techniques with regard to development time, maintenance time and robustness. Furthermore, we suggest that future studies in the field should assess the benefits and drawbacks of combining the CC&R and VGT technique. Suggestively by replicating our experiment and evaluating the combination of the techniques in terms of development time, maintenance time and robustness. This would allow an interesting comparison to our findings, as our findings show that the techniques have different benefits and drawbacks in different contexts, which indicates that a combination of the techniques may be beneficial.

## 6.5 Tool discussion

We are aware that the versions of the tools we used can be updated and thereby eliminating or at least mitigating their problems and defects. However, we still would like to make a contribution to the discussion about tool problems with our personal reflections and experience of the tools.

### UFT 11.52

The UFT tool that represented the VGT technique froze, i.e stopped responding, a lot when using Insight Objects stored in the object repository. We speculate that the freezes started to occur more frequently when the object repositories got too big, i.e. when a repository reached around 100 MB. Our solution to this problem was to create

one small object repository for each test case, instead of having one huge repository for all test cases. This lowered the frequency of crashes but did not totally solve the problem. Furthermore, we speculate that UFT was more likely to crash when adding objects manually to the object repository. Therefore we started to use the record and replay feature instead as this seemed to also lower the amount of freezes in the tool.

Moreover, UFT caused system exceptions when we used Insight Objects directly in actions, i.e. directly in the test scripts. However, when accepting the system exception by manually pressing the OK button on the pop-up window that appeared, the test case continued to execute as intended. To avoid getting system exception pop-ups, we used the Insight Objects in functions, stored in a separate function library.

Recent studies [6] have shown that the image recognition functionality of Sikuli, a tool that supports the VGT technique, could perceivably fail at random during test script development. The researchers that conducted the study explain that the failures could be because the image recognition's similarity level for the sought image was either too high or too low for the image recognition algorithm in use [6]. This is in our experience also true for UFT. The Insight object recognition could in some cases find several component matches in SystemX's GUI for one Insight Object in the repository. However, we do not consider this as a major problem as it could be easily fixed by increasing the similarity level of the Insight Object to higher than the standard of 80% or by manually modifying the Insight Object's captured image, so it captures as unique graphics as possible. We believe that the UFT tool is still immature. However, besides these mentioned problems, we are content with the tool's overall usability, features, layout and performance.

Furthermore, because UFT also supports all functionality that QTP provides, we suggest UFT as being a suitable tool to use in future research that evaluates a mix of the both techniques CC&R and VGT.

### **QTP 11.00**

We are very satisfied with QTP's usability and functionality. All interactions with the tool went smoothly, with only a few crashes at random occasions. We speculate that these crashes occurred due to the operation system in use as we could not find any way to provoke crashes of the tool. We did not experience any problems with the tool's object repository. We interacted with the repository without problems, even though it stored all test cases' objects. Furthermore, the tool's interface was straight forward and easy to use. All in all we are very satisfied with the tool's overall usability, features, layout and performance and favour working with the more mature tool QTP over UFT.

# 7

## Validity evaluation

This section presents the threats to validity of this thesis results. According to Wohlin et al. [28] there exists four main categories of threats to result validity; internal validity, external validity, construct validity and conclusion validity. The following section will discuss what threats that were identified, what effect they are perceived to have for the results and what steps that were taken to mitigate their effects.

### 7.1 Internal validity

Threats to internal validity are concerned with external influences, such as maturity of the subject and social influences, that can affect the independent variables [28]. There were two types of threats to internal validity identified for this thesis, i.e. Maturation and Selection which will be described below.

Maturation concerns how the subjects can react differently as the study, in this case quasi-experiment, progresses [28]. For example, the subjects might get bored of the work if it is uninteresting to them, which might affect the result or data collection negatively, e.g. by resulting in sloppy work or halting the study progress. Another maturation concern is that the subjects might learn the studied concepts at different rate, which can result in one subject being more efficient than the other.

In this thesis a systematic approach was created to reduce the learning effects of the subjects by switching tools after developing a test case, as presented in the randomization subsection under the Experimental design subsection 4.2.5. Furthermore, since the authors of this thesis were the subjects in the study, the risk of the subjects getting bored is considered minimal since they had a personal interest in acquiring valid and get reliable results.

Selection regards the selection of subjects for the study, or in this case quasi-experiment, and how representative they are for the population of possible subjects [28]. The selection threat was identified early in the planning phase of the industrial study, since the

study would only be conducted by two subjects that were chosen through convenient sampling. However, the subjects were still perceived to represent industrial practitioners that have limited knowledge of automated GUI based testing, e.g. testers/developers at DepartmentX that have not used QTP or UFT before. To raise the representativeness, the subjects prepared for the study by doing tool inspections of the testing tools and attended an educational workshop to increase their domain knowledge about SystemX. However, according to the guidelines set by Wohlin et al. [28] the study is still classified as a study performed by/with students rather than professionals.

## 7.2 External validity

Threats to external validity are concerned with how generalizable the findings are from a study, or quasi-experiment in this case [28]. Two threats were identified for the external validity of this work; Interaction of selection and treatment and Interaction of settings and treatment, which will be described below.

Interaction of selection and treatment refers to the effects of not having suitable subjects that are representative of the population we want to generalize the results for [28]. This is a threat to the thesis results since the subjects' experience of automated GUI based testing and domain knowledge was limited prior to the study. These knowledge limitations limit the perceived generalizability of the results for the desired population. However, the threat was mitigated as the subjects attended educational workshops and conducted tool inspections to raise their representativeness of industrial practitioners. As such, the subjects are perceived to represent at least testers and developers with limited experience, that work at CompanyX. However, according to the guidelines set by Wohlin et al. [28] the study is still categorized as a student study.

In turn, interaction of settings and treatment is caused by the studied phenomenon, SystemX, QTP and UFT in this case, not being representative in general industrial practice [28]. Thus, it is possible that the characteristics of SystemX, QTP and UFT or CompanyX's context are too specific to generalize the acquired results for other systems, GUI based tools or contexts. To mitigate this threat, the context of the study has been described in as much detail as possible within the thesis.

In addition, QTP and UFT are not the only CC&R and VGT tools available on the market and their representativeness for other tools is unknown. However, it is perceived that all CC&R and VGT tools share common characteristics, as also reported in related work for VGT [15], which supports the external validity of the reported results. However, because of this threat we suggest that future research should compare CC&R and VGT with different or more tools in order to verify our findings.

## 7.3 Construct validity

Threats to construct validity concern the design of the study, or quasi-experiment in this case, and how the results are connected to the theoretical background of the study [28].

There were two types of threats to construct validity identified for the thesis results; Mono-method bias and Experimenter expectancies, which are described below.

There exist a threat regarding mono-method bias [28] for the quantitative results presented in this thesis since the quasi-experiment was the only method that was used to obtain said results. However, since the quantitative data was collected through a systematic and rigorous methodology, the mono-method bias is perceived as low.

The second threat to construct validity relates to experimenter expectancies [28], which can cause the experimenter to be positively biased towards one of or the other of the studied phenomena/treatments/etc. However, since the authors did not favor either technique prior to the thesis, nor had any personal gain in the outcome of the study, this threat is considered limited.

## 7.4 Conclusion validity

Threats to conclusion validity concern the reliability of the results from a study, or the quasi-experiment in this case, and if correct conclusions can be drawn from the results [28].

Several steps were taken in the planning phase of the thesis work to reduce the threats regarding conclusion validity. Firstly, a systematic approach was created for data collection that the subjects strictly followed during the study in order to ensure that the collected data was consistent and coherent, regardless of who collected the information. Secondly, since the script development was performed by reusing functions, the subjects could ensure that the developed test cases had the same structure and functionality. These test case properties are perceived to have had a positive effect on the data collection and the consistency of the collected data.

# 8

## Conclusion

**T**HIS THESIS WORK presents a comparative study, with the established Component-based Capture & Replay (CC&R) technique and the novel Visual GUI testing (VGT) technique, with the purpose of evaluating which technique is perceived the most beneficial in terms of development time, maintenance time and robustness [6, 7, 15]. The study was performed as a quasi-experiment at CompanyX with an industrial system for the avionics domain. Analysis of the acquired results showed that VGT had lower development time than CC&R. Whilst CC&R had a lower maintenance time and higher robustness than VGT. Thus, these findings from the comparative quasi-experiment resulted in three key contributions:

- C1. The development costs, measured as time, associated with VGT are lower than for CC&R.
- C2. There exists no statistical significant difference in maintenance costs, measured as time, associated with CC&R and VGT. However, the CC&R technique was perceived to be better suited within the industrial context, because it had lower total maintenance cost.
- C3. CC&R is generally more robust than VGT.

These contributions provide value to the currently limited body of knowledge on VGT but also automated GUI based testing in general. Especially since automated testing has been proposed as a way of mitigating the problems of high cost, tediousness and error proneness, associated with manual test practices [4, 5]. However, support for automated high-level testing is still limited as CC&R, the currently most common automated GUI based testing technique in practice [1, 7], suffers from various limitations that affect its usability, cost, robustness and foremost maintainability [2, 10, 11, 12, 13, 14]. VGT is a novel GUI based test technique that is emerging in industrial practice that is perceived to have characteristics that can overcome the limitations experienced with CC&R [9].

However, VGT's body of knowledge is currently limited and in order to make any conclusion regarding the technique's long-term industrial applicability, additional research is required, e.g. research regarding the technique's robustness and maintenance costs [6].

Based on our findings we recommend companies with similar industrial context as CompanyX to (a) use CC&R whenever applicable but (b) complement their CC&R use with VGT for automated GUI based tests where CC&R is not applicable. In addition, because the techniques were found to have different benefits and drawbacks, we propose that a combination of the techniques can be the most beneficial in general practice and is therefore an important subject of future research.

In summary, this thesis contributes with (1) empirical results about the differences between CC&R and VGT in terms of development time, maintenance time and robustness, (2) evidence that both CC&R and VGT are applicable techniques in practice for GUI-based testing and (3) a general contribution to the body of knowledge on automated testing with novel results from a comparative industrial study of the CC&R and VGT techniques.



# Bibliography

- [1] A. M. Memon, Gui testing: Pitfalls and process, *Computer* 35 (8) (2002) 87–88.
- [2] A. M. Memon, M. L. Soffa, Regression testing of guis, in: *ACM SIGSOFT Software Engineering Notes*, Vol. 28, ACM, 2003, pp. 118–127.
- [3] E. Borjesson, 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.
- [4] M. Grechanik, Q. Xie, C. Fu, 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.
- [5] M. Grechanik, Q. Xie, C. Fu, Maintaining and evolving gui-directed test scripts, in: *Software Engineering (ICSE)*, 2009. IEEE 31st International Conference on, IEEE, 2009, pp. 408–418.
- [6] E. Alégroth, R. Feldt, L. Ryrholm, Visual gui testing in practice: challenges, problems and limitations, *Empirical Software Engineering* (2014) 1–51.
- [7] E. Alégroth, R. Feldt, H. H. Olsson, Transitioning manual system test suites to automated testing: An industrial case study, in: *Software Testing, Verification and Validation (ICST)*, 2013 IEEE Sixth International Conference on, IEEE, 2013, pp. 56–65.
- [8] E. Sjösten-Andersson, L. Pareto, Costs and benefits of structure-aware capture/replay tools, *SERPS'06* (2006) 3.
- [9] E. Alégroth, On the industrial applicability of visual gui testing, *Licentiate thesis, Chalmers University of Technology* (2013).
- [10] K. Li, M. Wu, *Effective GUI testing automation: Developing an automated GUI testing tool*, Wiley. com, 2006.

- [11] E. Horowitz, Z. Singhera, Graphical user interface testing, Technical report Us C-C S-93-5 4 (8).
- [12] M. Grechanik, Q. Xie, C. Fu, Experimental assessment of manual versus tool-based maintenance of gui-directed test scripts, in: Software Maintenance (ICSM), 2009. IEEE International Conference on, IEEE, 2009, pp. 9–18.
- [13] F. Zaraket, W. Masri, M. Adam, D. Hammoud, R. Hamzeh, R. Farhat, E. Khamissi, 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. 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, 2001, pp. 114–117.
- [15] E. Alegroth, M. Nass, H. H. Olsson, Jautomate: a tool for system-and acceptance-test automation, in: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, IEEE, 2013, pp. 439–446.
- [16] G. J. Myers, C. Sandler, T. Badgett, The art of software testing, John Wiley & Sons, 2011.
- [17] K. Pohl, C. Rupp, Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam-Foundation Level-IREB compliant, " O'Reilly Media, Inc.", 2011.
- [18] S. Berner, R. Weber, R. K. Keller, Observations and lessons learned from automated testing, in: Proceedings of the 27th international conference on Software engineering, ACM, 2005, pp. 571–579.
- [19] A. M. Memon, A comprehensive framework for testing graphical user interfaces, Ph.D. thesis, University of Pittsburgh (2001).
- [20] B. N. Nguyen, B. Robbins, I. Banerjee, A. Memon, Guitar: an innovative tool for automated testing of gui-driven software, Automated Software Engineering (2013) 1–41.
- [21] W.-K. Chen, T.-H. Tsai, H.-H. Chao, Integration of specification-based and cr-based approaches for gui testing, in: Advanced Information Networking and Applications (AINA), 2005. 19th International Conference on, Vol. 1, IEEE, 2005, pp. 967–972.
- [22] A. Ulrich, A. Petrenko, Formal approaches to software testing, in: Third International Workshop on Formal Approaches to Testing of Software (FATES), Springer, 2003, pp. 1–14.

- [23] T. Lalwani, S. N. Kanoujia, T. Howarth, M. Smith, QuickTest Professional Unplugged, KnowledgeInbox, 2011.
- [24] T.-H. Chang, T. Yeh, R. C. Miller, Gui testing using computer vision, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2010, pp. 1535–1544.
- [25] L. Hewlett-Packard Development Company, A guide to image-based testing with hp uft’s “insight” (Februari 2014).  
URL [http://www.automation-consultants.com/products-Unified\\_Functional\\_Testing-135](http://www.automation-consultants.com/products-Unified_Functional_Testing-135)
- [26] T. Lalwani, M. Garg, C. Burmaan, Uft/QtP Interview Unplugged: And I Thought I Knew Uft!, KnowledgeInbox, 2013.
- [27] Scrum.org, The home of scrum (April 2014).  
URL <https://www.scrum.org/>
- [28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer, 2012.
- [29] R. Wilcoxon, Modern statistics for the social and behavioral sciences: A practical introduction, CRC Press, 2011.
- [30] J. P. Kleijnen, Verification and validation of simulation models, European Journal of Operational Research 82 (1) (1995) 145–162.