FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Comparative Study of GUI Testing Aproaches

Rui Carvalho

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Paiva (PhD)

July 7, 2016

A Comparative Study of GUI Testing Aproaches

Rui Carvalho

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: João Carlos Pascoal Faria

External Examiner: João Miguel Fernandes

Supervisor: Ana Cristina Ramada Paiva

July 7, 2016

Abstract

Most of the modern software applications feature a Graphical User Interface (GUI), which turns the application easier to use, promoting higher productivity and better accessibility, and offering flexibility in how users perform tasks. However, due to GUI's complexity, the GUI testing process can be a time-consuming and intensive process. Therefore, automate the process as much as possible is indispensable to test any more evolved graphic user interface.

There are some common automated GUI testing approaches, but while most of them require substantial manual efforts, others lack reusability or are only able to find specific types of errors. Many researchers state that a variety of techniques should be used.

A new model-based testing approach, called Pattern- Based GUI Testing, was implemented in order to increase systematization, reusability and diminish the effort in modelling and testing. It is based on the concept of User Interface Test Patterns (UITP), which contain generic test strategies for testing common recurrent behavior (UI Patterns) on GUIs, and supported by the PBGT Tool which provides an integrated modeling and testing environment that supports the crafting of test models based on UI Test Patterns, using a GUI modeling DSL (PARADIGM).

As a novel proposal, it is entirely relevant to submit it to systematized experiments and tests in order to assess its good performance/behavior and compare it with other techniques. Thus, this dissertation work mainly addresses PBGT's approach, aiming to compare it with different testing approaches/tools in what concerns to errors/fault detection, ease of use, and overall efforts required to test the application.

To perform the experiments, mutations were introduced in each of three different web applications - iAddressBook, TaskFreak and Tudu - to cover a greater number of use cases, and each mutant was tested by each of the selected or developed testing tools which implement the considered approaches. Those approaches' benefits and problems are then conveniently described.

The experiment's results showed that the PBGT and the Capture/Replay approaches required a similar time to build the scripts/models; however, the former was able to find more faults but taking longer to kill a mutant. On the other side, the random testing approach was the less capable of detecting faults, while being one of the most time consuming ones.

Resumo

A maioria das aplicações de software modernas apresentam uma Interface Gráfica de Utilizador (GUI), que torna a aplicação mais simples de usar, promovendo maior produtividade e melhor acessibilidade, e oferecendo flexibilidade na forma como os utilizadores podem executar tarefas. No entanto, devido à complexidade das GUIs, o processo de teste de GUI pode ser moroso e intensivo. Assim, automatizar o processo tanto quanto possível é indispensável para testar qualquer interface gráfica mais complexa e evoluída.

Existem diferentes abordagens para testes automatizados de software através da sua GUI, no entanto, enquanto algumas requerem esforços manuais substanciais, outras apenas são capazes de encontrar erros específicos ou não permitem a reutilização de casos de teste após alterações de sistema ou GUI. Muitos investigadores afirmam que devem ser utilizadas diferentes técnicas/abordagens para um bom processo de teste.

Uma nova abordagem baseada em modelos, denominada de Pattern-Based GUI Testing, foi implementada a fim de aumentar a sistematização, reutilização e diminuir o esforço da modelação e teste de GUIs. Baseia-se no conceito de Padrões de Teste de Interface de Utilizador (UITP), que contêm estratégias de teste genéricas para testar características recorrentes e comuns (UI Patterns) em GUIs. É apoiada pela ferramenta PBGT, que integra um ambiente de modelação e execução de testes de modo a suportar a criação de modelos de teste com base em UITPs, com recurso a uma linguagem específica de domínio (PARADIGM) para modelação da GUI.

Como a abordagem é recente, é relevante submetê-la a experiências e testes sistematizados a fim de avaliar o seu bom desempenho/comportamento e compará-la com outras técnicas. Assim, este trabalho de dissertação baseia-se na avaliação e comparação da abordagem PBGT em relação a outras ferramentas e técnicas, no que diz respeito à detecção de falhas, facilidade de utilização, e aos esforços necessários para testar a aplicação.

Para a realização de experiências, foram introduzidas mutações em três aplicações web diferentes - iAddressBook , TaskFreak e Tudu - de modo a abranger um maior número de casos de uso, e cada mutante foi, por sua vez, testado por cada uma das ferramentas selecionadas ou desenvolvidas e que implementam as diferentes abordagens de teste consideradas.

Os resultados mostraram que as abordagens de teste PBGT e *Capture/Replay* apresentaram resultados muitos próximos no que concerne ao tempo necessário para criar

modelos e gerar scripts de teste, respetivamente. No entanto, a abordagem PBGT foi mais capaz de detetar uma maior quantidade falhas, ainda que requirindo mais tempo para o fazer. A abordagem de teste *Random* foi a que mostrou ser menos eficaz a nível de deteção de falhas, bem como uma das que mais tempo requer para todo o processo.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Professor Ana Paiva, for the continuous attention, support, and guidance during the whole process of this work. Her encouragement and knowledge were indispensable to keep everything on track.

I would also like to thank my parents and my brother for all the kindness and support, making this work much easier.

Last but not least, I thank everyone that contributed to my academic education.

Contents

List of Figures

List of Tables

Abbreviatons

API	Application Programming Interface
DOM	Document Object Model
DSL	Domain Specific Language
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
PBGT	Pattern-based GUI Testing
PHP	PHP Hypertext Preprocessor
SRS	Software Requirement Specification
SUT	Software Under Test
UI	User Interface
XML	eXtensible Markup Language
XPath	XML Path Language

Chapter 1

Introduction

This chapter presents the description of the context and motivations which led to the proposed dissertation's theme. Also, it provides a succinct description of some software testing concepts mentioned along this document.

1.1 Context

1.1.1 Software Systems and GUIs

Nowadays, in a modern society, software systems are present everywhere and in a wide range of domains. From bank and payment systems to smartphone applications, it is broadly used by a multitude of people directly or indirectly. Being the implementation and adoption of such software systems highly expressive, and taking into account its continuous growth, it is possible to say that our daily lives are already dependent on their proper functioning [1]. As Marc Andreessen said [2], "software is eating the world".

To provide the users with a simpler and more flexible way of interaction with software systems, the vast majority of today's software applications feature a Graphical User Interface (GUI). GUIs display relevant information and possible actions to users through virtual objects, or widgets, which are graphic elements such as buttons, text or edit boxes that make it intuitive to use software [3]–[5]. Software developers are increasingly dedicating a substantial quantity of the code to implement the GUI, being stated by Myers and Rosson [6] that, according to a survey they conducted, an average of 48% of the overall application source code is used for implementing a graphical user interface in order to release a user-friendly software.

The Graphic User Interface can be defined as a front-end for the subjacent component of the system. Thereby, the user interaction with the software is established by performing sequences of events, such as type-in text, click buttons, select menus and menu items, over GUI's graphic elements. In its turn, the GUI layer sends these events' information to the abovementioned subjacent software component via method calls or messages [3], [7], [8]. These events change the software state which, as said by Nguyen et al. [7], "may or may not include a change to the visible GUI itself".

It is common ground that the accuracy of the functionality of a software is considerably dependent and related to the tests performed to the system before it is public released. Having been said that GUIs are the connection establishers between the software and its end users, being the only way they have to access the functionality of the software, it makes it attractive for testers to perform a GUI level test because it means testing a program's correct behavior from the user's perspective. [9]

These consist of end-to-end tests of an application through its graphical interface that, according to Memon [3], "help ensure safety, robustness and ease of use of the entire software system".

1.1.2 GUI Testing

GUI testing consists in the execution of events associated to widgets and in the monitorization of the resulting changes in the program state. Thereby, the designed test cases are modeled as a sequence of user events, being defects expressed as failures observed through the GUI. According to Brooks et al. [8], these failures can occur as a consequence of "GUI defects" or "business logic defects". The former can be defects such as "incorrect label text" or "buttons missing a caption", and the latter is associated to incorrect computing result values. Not only the code associated to the GUI itself is tested, allowing us to define GUI testing as a process of testing a software application for functional correctness through its GUI [7], [8].

Due to the complexity of the modern GUIs, the testing process is a time consuming and intensive task. Therefore, automate the process as much as possible is indispensable to test any more evolved graphic user interface in order to make it more reliable, less time consuming and less costly. However, automated GUI testing is a much more difficult task than API test automation or conventional software testing [1]. Users can trigger events at almost any time by performing actions, and can perform different actions to achieve the same goal. In order to cover all the possibilities, an automatic test suite has somehow to simulate the possible sequences of events, but the space of all the possibilities can be enormous. Also, observing the state of the GUI is very difficult because it is a combination of all of its components' states. If the GUI state is invisible, observing it is almost impossible [10].

The scientific and professional community defends the idea that it is necessary to use a variety of techniques for effective GUI testing [7]. Several kinds of testing tools, based on different GUI testing techniques or approaches, have been developed. The spectrum of tools goes from these that can only automatically run test cases to those that have the ability to almost automatize the entire testing process, constructing a GUI model by a reverse engineering process, generating and running test cases and oracles and verifying results [11]. However, the majority of the currently available tools for GUI testing are somehow limited in what concerns to the techniques they can apply.

1.2 Automated GUI Testing Concepts

Here we present the definition of some concepts, inherent to automated software testing, used throughout this dissertation report.

- **Black-box technique** – The tester views the software as a black-box, only having information about the test inputs and expected outputs. He does not need to have any knowledge regarding the underlying code layer. The way the program got to provide the output is not important. The code is never inspected and the tester only needs to have knowledge about the program's specifications. Techniques such as random testing and typical model-based testing consist in black-box testing. [12]

- **Reverse Engineering Approach** – Models representing the GUI are directly extracted by this technique from the binaries of the application under test. [13]

- **Test Oracle** – According to Memon et al. " [14], a test oracle is a source of information that "determines whether a software executed correctly for a test case (...)". It contains "information that represents expected output, and an oracle procedure that compares the oracle information with the actual output".

- **Test Suite** − Test suite is a container that has a set of tests. Given that the successful completion of one test case must occur before the beginning of the next one, gaps in a testing session can be identified [15].

1.3 Motivations and Objectives

Considered a validation process, software testing is used to check if a software application is functioning properly and in line with the specified requirements, ensuring that it meets the desired quality level; also, "it is performed to uncover and correct as many potential errors as possible" [16]. Automated testing tools are designed to simplify the tester's job, helping him in different phases of the testing process depending on the chosen technique, as will be seen in the following section. However, not all tools and testing approaches are equal. As a matter of fact, they differ significantly from each other. Many new approaches and tools are constantly emerging, so their effectiveness in defects detection should be evaluated before they achieve wide acceptance. A common way to assess test techniques is through empirical studies, in which different software techniques are experimented with known defects.

Howden's [17] proposed definition of reliable test set states that "the success of a reliable test set implies the program correctness", as simplified by Chen, Tse and Zhou [18]. Thus, and based on this, we can infer that a totally reliable testing tool can find that an incorrect program is actually not correct. Moreover, we can say that, considering a set of tools testing a program with known faults, the higher the number of detected faults by a certain tool is, the greater that tool's reliability/effectiveness will be.

A good testing tool/approach and a good set of test cases should detect different type of failures or errors in order to be a worthy holder of the adjective. There are many types of test automation techniques. Some commonly mentioned and used approaches are the Unit Testing, Capture/Replay Testing, Random Testing and Model-based Testing.

In this work, we will make use of the functional testing of a software via its GUI, recurring to the abovementioned approaches, in order to perform experiments that will allow us to evaluate the efficiency and effectiveness of different GUI testing approaches and GUI testing tools. Thereby, different approaches and different GUI testing tools will be used to test both supposedly correct programs and mutants of these same programs. Mutations will then be applied to the study subjects, meaning that they must consist in open source applications.

A model-based GUI testing approach, the Pattern-Based GUI Testing (PBGT), was recently proposed by Ana Paiva in order to simplify the typical model-based process. As a novel proposal, it is entirely relevant to submit it to systematized experiments and tests in order to assess its good performance/behavior and to compare it with other techniques.

Thus, the main goals of this dissertation's work are then established as:

- Compare the modeling and configuring efforts of the PGBT's DSL modeling language (PARADIGM) with other GUI testing approaches;

- Compare different testing approaches/tools in terms of detected faults.

1.4 Structure of the Report

This document is structured into four main chapters. The first (and current) chapter, "Introduction", sets the scene for the main body of the study. The theme and context that underpin this study are introduced, and the motivations and goals are explained in detail. Some concepts regarding the theme are also addressed. Chapter 2 presents the state-of-the-art on GUI testing approaches and GUI testing tools, divided into separate subsections. A clarifying description of the scientific experiment method, as well as of some of the necessary background concepts is also provided. Chapter 3, "The Experiment", describes each step inherent to the process of our investigation in detail. The results and a respective discussion of them is presented. Finally, Chapter 4, "Conclusions and Future Work", overviews this research work in terms of achieved goals, results, and limitations of the research that were identified in the overall process.

Chapter 2

State-of-the-Art

This chapter provides a state-of-the-art in what concerns to GUI testing approaches/techniques, as well as a brief description of the most used/popular GUI testing tools.

The first section, "GUI Testing Approaches," covers up the main testing techniques and the PBGT approach is also generally addressed, whilst the second one, “GUI Testing Tools”, present the different testing tools based on those specific testing methodologies.

2.1 GUI Testing Approaches

This section presents an overview of the most common GUI testing approaches, according to [11], and a new one – Pattern-Based GUI testing approach - which we will address in the scope of this study. Other approaches are mainly academic and so were not considered.

2.1.1 Unit Testing

Unit Testing is a technique supported by semi-automatic tools and it is performed on methods or components level, a lower level of system abstraction. When recurring to this approach to perform GUI testing, the GUI unit test cases, consisting of class/method calls to simulate the users’ interaction (sequences of actions) with different GUI elements/objects, must be programmed manually [3], [19].

These kind of tools/frameworks are only useful to organize and execute test cases. Some of them also include specific libraries with the aim of simplifying the test cases creation and observation of the virtual GUI objects' state. The procedure is similar to the API testing [11].

Assertions are inserted in the test cases to verify if the classes/methods of the unit under test are properly functioning. So, during the execution, it is checked if the resulting output matches the expected result [20]. However, it is difficult to create GUI assertions for testing, since the output commonly consists of the generation of a bitmap, which can be stochastic [19].

Börjesson et. Feldt [21] mentioned that, owing to the fact that Unit Testing is a technique established to operate on a lower level of system abstraction, different studies discuss the applicability of these techniques on high-level tests. They also state that this uncertainty resulted in the development of explicit automated testing techniques for system tests such as the case of Capture/Replay, which will be discussed nextly.

2.1.2 Capture/Replay Testing

Capture/Replay is a software testing technique, supported by semi-automated tools, whereby manual interactions with the GUI components of the system under test (SUT) are recorded in order to generate automated test scripts that can posteriorly be replayed [21].

Thus, tools that support this technique allow its users (the testers) to create, execute and verify the results of automated tests for systems that feature a graphical interface. As the name implies, its main functions are capturing/storing manual interactions of the tester with the GUI to automatically replay them later.

The first phase, the user session capturing, consists in registering all the objects displayed on the SUT and recording the entire sequence of the user's manual interactions with these objects in a test script file until the recording mode is stopped [14], [22]. The user interaction can be captured via direct references to GUI's components, on the GUI component level, or via coordinates for the location of the GUI component of the software under test, on the GUI bitmap GUI level [21].

The generated file will contain a registry of all the actions performed, including mouse movements, chosen options, entered inputs and the obtained result. This way, this method allows testers to easily select the widgets and their properties that they want to test/evaluate afterwards [22]. A user session is captured as a test case.

In the second phase, the recorded scripts, which are written in a language adopted by the tool, can be repeatedly executed or, in other words, automatically replayed on the GUI at any time. Thus, the execution of a non-edited script will be an exact replay of the actions performed during the capture session. The results obtained with the capturing sessions can be considered the test's expected outcome, being the states registered used as oracle information, which means that they will be compared with the results obtained during posterior automatic executions as a way to determine the final test result [22]. If any mismatches occur, which is to say that the SUT doesn't behave as expected or if checkpoints are violated during a replay, the test will fail and the tool will report them as application's defects. It should be noted that sometimes the file script needs to be edited and debugged by the tester so that the proper testing is completed [23].

It is frequent that the detected defects are not inherent to SUT faults itself but more related to the incapacity that some Capture/Replay tools have to deal with GUI component changes, such as API or GUI layout changes [21]. Memon, Pollack et Soffa [24] pointed out that testers employing these tools generally work with a small number of test cases, or small modular scripts.

Similar to Unit Testing, the tests designing is manual and Capture/Replay tools only automatize the capture and execution of test cases. Yuan et al. [20] stated that the domain of GUI states explored by the captured test cases is correlated to the to the quality of the recorded sessions and testers competences.

Moreover, the use of this technique has some drawbacks and it oftentimes turns to be bothersome. Testers can be constantly interrupted to insert checkpoints, making the process laborious and tedious. Another of the peculiarities is that the script is only generated when the user session is properly completed without errors, which means that those errors will have to be iteratively corrected to make it possible to complete the recording session in order to get the script. Consequently, when the script is obtained the errors were already found. It can then be said that this technique is more useful to assist regression testing [23].

Also, the fact that a change in the software's GUI frequently means a need to change the test scripts, as mentioned, makes the use of these technique an expensive option due to the high maintenance costs associated.

2.1.3 Random/Monkey Testing

The Random Testing technique, also known as monkey testing, is a black-box testing technique and can be the simplest test strategy from all of the referred. It does not intend to automatically run test cases but to automatically simulate and run possible user actions. In general, it consists in randomly triggering various GUI events of the SUT without any knowledge on how to use the application, which means without any "typical user" bias, by the emission of mouse clicks and key strokes, in order to crash or freeze the GUI/software. At each test run, the technique develops different test cases and explores the software under test in a new way [25], [26].

They are supported by fully automated testing tools, commonly and metaphorically called "test monkeys", a name derived from the aphorism: "A thousand monkeys typing on a thousand typewriters will eventually type the entire works of William Shakespeare", because in very little they rely on human knowledge or, in this case, in his understanding of how to use the application, but can generate relevant test cases.

Random Testing tools that can be distinguished into two main types, according to the methodology implemented, the smart and dumb monkeys.

The former has a general knowledge of how to access GUIs but not any specific knowledge regarding the application. Their behavior is based on completely random clicks and keystrokes and they can not recognize a bug when in front of one. Dumb monkeys are used to detect memory leaks, access violations and program hang-ups [27].

The latter, smart monkeys, have some specific knowledge of the application and can generate random actions based on it. That knowledge can come from a state table or from a model of the application that guides and helps them to make decisions about actions to perform. They can find relevant bugs [28].

Software testers claim that test monkeys can often find serious bugs that can be caused by "dark" sequences that are difficult to find, even for experienced testers. On the other side, and according to Yang [26], a number of researchers call into question the effectiveness and usefulness of such tests since they are unconcerned about the knowledge of software specification and the subjacent software structure.

One of the referred problems relates to the fact that these tools may have to execute for a long time until they can cause a crash in the application, resulting in long sequences of events in an absolutely random distribution.

The second relates to the difficulty of reliably reproduce these sequences, in order to obtain information about what caused the software to crash, given that usually the tester has to inspect the output of the debugger where the application runs [25].

However, Yang [26] states that are also studies such as [29] claiming that, after experiments, Random Testing performed better than other strategies.

The fact that those tools work completely automatically allows some savings on testing costs, being sometimes an attractive factor. Also, it is recurrent to see test monkeys used in addition to other techniques [25].

2.1.4 Model-based GUI Testing

In a model-based GUI testing approach, the tester should at first create or generate a model, according to the DSL implemented by the adopted testing tool, that should closely represent the structure and behavior of the SUT'S GUI. Sometimes, testers choose to automatically generate the models with the appliance of reverse engineering techniques. When crafted manually, testers usually derive the GUI models from the requirements specification.

Most of the models can be defined as an "eventflow graph", reason why it is stated that the detection of faults highly depends on the tester capacity to conveniently define all the possible GUI states in the crafted model. The derived test cases/scripts are then executed and the produced output it compared to the expected one.

In recent approaches, the graph nodes tend to represent GUI events, while edges represent de relation between these events [20], [30].

2.1.5 Pattern-based GUI Testing

Pattern-based GUI testing is a new model-based testing approach, based on the concept of UI Test Patterns which feature a set of test strategies that simplify the test of UI Patterns on GUIs. UI Patterns can be defined as common characteristics applied to most of the modern GUIs.

Those patterns can be behavioral elements, such as login, sort, find, etc., or structural elements, such as forms or group [31].

Input, for example is a pattern used to test the behavior of input fields, when valid and invalid input data is submitted. In its turn, login pattern is used to check if it is only possible to authenticate with valid login credentials (username/password).

Figure 1 shows their representation in PBGT tool.

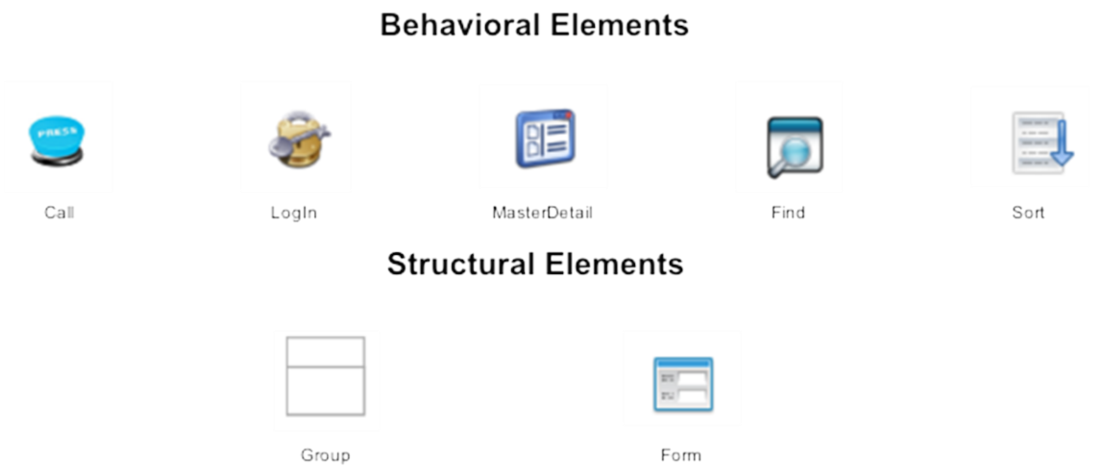

Figure 1: UI Patterns.

Those elements, representing UI Patterns, can be drag-and-dropped in a modelling environment, called PARADIGM-ME, in order to easily model the GUI of the SUT using a domain specific language for GUI modeling, PARADIGM. The patterns can also be combined to obtain newer ones. The resultant crafted GUI models will have a certain level of abstraction, leading its derived tests to be abstract versions of executable tests as well.

To locate and identify the SUT’s elements, PBGT Tool relies on Selenium WebDriver libraries, as well as on Sikuli, to locate them. When Selenium cannot identify an element, Sikuli tries to via image recognition (given that images of elements can be collected when mapping elements).

Besides the modeling, PARADIGM-ME also supports the automatic test case generation and execution. In test case creation, different parameters can be customized, and different strategies lead to different results. Random strategy, for exemple, could generate an enormous number of test cases, since it randomly considers all the possible paths.

PBGT Tool is freely available as an Eclipse plugin and allows the GUI testing of web-based and Android applications [32], [33].

2.1.5.1 PBGT Technical Features and Advantages

Many reasons are pointed by the research team [32] to encourage the adoption of PGBT approach and tool.

The first one is related to reusability, since UI Test Patterns can be reused during the GUI modeling and testing process.

Regarding GUI modeling, it is comparatively easier than other modeling approaches, since the drag-and-drop GUI of UI Patterns turns the process more intuitive and less complex with no coding involved. It is also stated that users even users with little knowledge on the subject can start modeling and testing software in short time. The GUI models are created or generated without needing to access the SUT source code.

PGBT tool can be used to more easily model and test web-based and mobile (Android) applications, through their GUIs, and it also allows the integration of new UI Test Patterns, and the edition/adjust of the existing test strategies, in order to support new UI trends.

2.1.6 Alternative Approaches

With base on these testing approaches, and taking into account their limitations, new approaches are emerging as adaptations/extensions of each of them. Examples are the Visual GUI Testing, as an adaptation of the typical Capture/Replay technique, that can be defined as a "tool-driven test technique where image recognition is used to interact with, and assert, a system's behavior through its pictorial GUI as it is shown to the user in user-emulated, automated, system or acceptance tests" [34].

Zoltán Micskei had also identified a "Gap Between Academic Research and Industrial Practice in Software Testing" [35] and wrote a document containing an extensive list of papers. Those papers suggest new approaches regarding testing activities which he categorized in: test generation, regression testing and empirical evaluations of tests.

2.2 GUI Testing Tools Overview

As mentioned, testing automation tools are designed to make the tester's job easier and more efficient. Based on different testing approaches, tools differ from those that only simplify the test cases execution to those that have the ability to almost entirely automatize the testing process. However, and although the nature and type of test cases can vary in function of the used technique, they all explore the GUI state domain via sequences of GUI events.

The most frequently mentioned tools in the literature are the *eggPlant Functional* by TestPlant[1], *HP WinRunner*[2] by Mercury Interactive/HP, *Ranorex*[3] by Ranorex GmbH, IBM's *Rational Robot*[4] and IBM's *Rational Visual Test*[5], *BadBoy*[6] by BadBoy Software, Segue's *SilkTest*[7], among others. However, and since these are commercial tools, they will not be included in this work.

Regarding **Capture/Replay** testing tools, Prabhu et Malmurugan (Prabhu & Malmurugan, 2011) presented a detailed survey of some open source tools which they considered the most relevant "based on the events and fields needed for an ideal GUI testing tool". These exactly same GUI testing tools, namely *Abbot*, *Jacareto*, *Pounder*, *jfcUnit* and *Marathon*, were also subject of a comparative study of open source Capture/Replay conducted by Nedyalkova et Bernardino [36].

The comparative study [36] showed that *Jacareto* is the tool that provides better support to the capture/replay sessions and also that it is the easiest to learn and use. However, the comparison did not address the regression testing capabilities, which is the main goal of the technique, since the authors only aimed to identify the tool with better capture/replay performance and not requiring lots of efforts to learn how to use it. The mentioned tools are used to test applications with Java based GUIs.

Some less known Capture/Replay tools that also appear in some studies are *GUI crawler*, *Android GUITAR* and *RERAN*. These apply the technique in order to help the testing of Android applications through their GUI.

To perform the technique on web-based applications, *Sahi*[8], *WebTst*[9], *TestGen4Web*[10] and Simple Web Automation Toolkit (*SWAT*[11]) are open source tools/frameworks that do so.

Regarding **model-based** GUI testing tools, *GUITAR*[12] (GUI-based apps), *Selenium* with *MISTA* (Java, C, C++, C#, HTML, and VB), *Jubula*[13] (Swing, SWT/RCP/GEF, JavaFX, HTML and iOS) and *Spec Explorer* are the main ones are in what concerns to the number of studies that mention them. There is also an empirical study, authored by Lelli, Blouin et Baudry [30], that proposes to compare *Jubula* to *GUITAR*.

[1] http://www.testplant.com/
[2] https://softwaresupport.hp.com/document/-/facetsearch/document/KM01033448
[3] http://www.ranorex.com
[4] http://www.ibm.com/support/entry/portal/product/rational/rational_robot
[5] http://www-03.ibm.com/software/products/pt/functionalTM
[6] http://www.badboy.com.au
[7] http://www.segue.com
[8] http://sahipro.com
[9] http://webtst.sourceforge.net
[10] https://sourceforge.net/projects/testgen4web
[11] https://sourceforge.net/projects/ulti-swat/
[12] https://sourceforge.net/projects/guitar/
[13] http://www.eclipse.org/jubula/

To perform a GUI testing session based on the **unit testing** technique, the tester has at his disposal a variety of frameworks that can be used to assist him to generate/write test cases. *Abbot*[14] (which integrates both Capture/Replay and unit testing) and *HtmlUnit*[15], for example, were designed with the purpose of being used within another testing framework such as *JUnit*. *Abbot* is used for Java GUI applications testing, while *HtmlUnit* aims to test web-based applications [37].

Other unit testing tools for web-based applications testing, such as *Selenium WebDriver*, *Canoo WebTest*, *JWebUnit*, *JSFUnit*, use *HtmlUnit* as the underlying "browser".

The number of studies that mention the use of the unit testing technique to perform a GUI test is comparatively lower.

In what concerns to **random testing**, the great majority of the recent studies refer this approach to test mobile applications. However, the authors don't provide access to their tools. Alégroth [38] have also stated that there exists a "gap for a high-level technique that can perform user emulated random testing through the SUT's GUI".

MonkeyRunner[16] is a popular mobile (Android) testing tool among recent studies regarding GUI Random testing tools, and it also supports unit testing. The tool makes it easier to write a Python program that installs, runs, sends keystrokes and takes screenshots of an Android application UI [39].

There also is a lesser known application, *Doit*[17], that according to its website is a "scripting tool and language for testing web applications that use forms". It has the capability of "generate random or sequenced form fill-in information, report results (into a database, file, or stdout), filter HTML results, and compare results to previous results" [40].

2.3 Some Tools Description

The following subsections briefly describe some of the mentioned tools for which little information was found, leading us to believe they are not very widespread.

2.3.1 WebTst

The *WebTst* is an easy to install and learn open source tool, proposed by Francisco Rosa. It allows testers to "either use prebuilt tests or to roll their own as needed" [41].
The tool implements a recorder component, allowing testers "to save all their actions via their usual client browser", and a management/playback component. With the management component, testers can easily "change recording items parameters" such as "test types or test

[14] http://abbot.sourceforge.net/doc/overview.shtml
[15] http://htmlunit.sourceforge.net
[16] http://developer.android.com/tools/help/monkeyrunner_concepts.html
[17] http://doit.sourceforge.net

parameters", delete items of the recorded session/script, "set tests to run daily for smoke tests", replay the "tests and check correct completion of all test items". The tool also "allows developers to build upon the out-of-the-box set of tests by hooking up their own tests to the existing ones via a well-defined interface".

2.3.2 Jubula

Jubula is a semi-automated model-based GUI testing tool for Swing, SWT, RCP, GEF and HTML based applications. It implements a code-free testing approach, providing a drag-and-drop test specification from predefined test libraries that contain modules including a comprehensive list of actions, allowing acceptance tests to be written from the user's perspective, seeing the software as a black-box. It is contributed by BREDEX GmbH[18] and it is based on a previous commercial tool, the *GUIdancer*.
It allows platform independent testing on Windows and Linux, implemented under a client-server architecture for distributed testing. The test multiple applications. It allows the test multiple applications, portability and version control via exports in XML format.

2.3.3 Sahi

Sahi is a Capture/Replay testing tool for web-based applications which supports Java and JavaScript. The generated script looks like JavaScript, but it is not executed as the regular JavaScript on the browser. The tool controller (IDE) can be used in various browsers. Sahi's APIs is not dependent on the HTML structure, since it does not use XPaths, featuring methods to help finding one element in relation to another that will work properly even if the structure of the page changes [42].

2.3.4 Abbot

Abbot is a GUI testing framework that can be used to generate/write test cases directly from Java code as well as to capture and replay scripts. It is an extension of JUnit extension for Java Swing/AWT GUI testing.

It provides methods that allow the tester to simulate user's actions and to inspect the GUI components state. As said, the framework can be called directly from the Java code or accessed without any code by using scripts. It integrates a visual test script editing tool (Costello) that allows the tester to easily access, explore and control an application, and to execute the XML or Java test scripts.

[18] http://www.bredex.de/home_en.html

In short words, *Abbot* is a framework supports Capture/Replay and Unit Testing. It can be used to create functional tests for existing GUI apps, and Unit tests for GUI components [43].

2.4 Mutation Testing

Adequacy metrics or *testing criteria* are used to evaluate test cases, being Mutation Analysis an established test criterion [44]. Mutation analysis involves systematic transformation of a program into very similar versions of it by the introduction of syntactical changes, and determines if tests can differentiate the mutated code from the original source code [45].

In Mutation analysis, a set of programs p' called mutants is generated by a few single syntactic changes to the original program p. The creation of these mutants is based on syntactic rules, called mutant operators, that change the syntax of the program by insertion, replacement, or deletion operators [45], [46].

In the next step, a test T is applied to the system. It is very important to successfully execute the test T against the original program p before starting the mutation analysis. Then, if the output of running p' is different from the output of running p for any test case in T, the mutant p' is said to be *killed*; if it is the same from the output of p, it is said to be have *survived*. After all test cases have been executed, there may still exist a few remaining surviving mutants [46]. There are two reasons for mutants to remain “alive” [47]:

- The test data that was generated is not able to distinguish the mutants, having to be modified until it kills all the mutants [47].
- The mutant p’, in spite of the modification of the test data, show the same output as the original program p. In this case, the mutant is said to be an equivalent mutant and it cannot be killed [47]. The difficulty of automatically detecting all equivalent mutants, even with the advances in mutation testing, has been a barrier that keeps Mutation Testing from being more used [46].

Mutation Testing concludes with the mutation score or adequacy score, which is defined as the ratio between the number of killed mutants and the number of non-equivalent mutants. The range of possible results for this score is between 0.0 and 1.0, being 1.0 the best possible score to obtain, meaning that this test set can kill all the non-equivalent mutants – in this case, the test is said to be 100% mutation adequate [48].

The power of this technique relies on the ability of the mutants to represent real faults [44]. However, it is impossible to generate mutants representing all of the potential faults. Thus, Mutation Testing usually targets only a subset of these faults, the ones that are really close to the original version of the program, hoping to simulate all the possible faults. This theory is based on two hypotheses: the Competent Programmer Hypothesis and the Coupling Effect [46]:

- The Competent Programmer Hypothesis states that programmers are competent, and so the versions of the programs that they develop tend to be very close to the correct version. Thus, only faults constructed from simple syntactical changes are considered, representing the faults that “competent programmers” would make [46].
- The Coupling Effect concerns the type of faults used in mutation analysis. It is believed that this effect operates in such a manner that a test set that kills first-order mutants (program variants with one fault apiece) would be able to kill higher-order mutants (program variants with more than one fault apiece) too. Many empirical studies support this hypothesis [49].

Although Mutation Testing is able to effectively evaluate the quality of a test set, it still suffers from a number of problems [46], [48]. One problem is the high computational cost, even with small and simple programs. Several techniques were developed to reduce this problem, such as selective mutation, mutant sampling, weak mutation, schema-based mutation and separate compilation [48]. The other problems are related to the amount of human effort involved in using Mutation Testing [46].

2.5 Scientific Method

The scientific method is as a logical, orderly way to answer a question or solve a certain problem [50]. Often, science textbooks describe it as if there is a strict, simple flow chart that all scientists follow, which is clearly an oversimplification [51]. In different ways, it has probably been used ever since people started questioning about the world around them [50].

Francis Bacon developed a method of scientific investigation, with the emphasis on the primary role of empirical observation and experimental testing of hypotheses, which was widely followed for centuries. Darwin, coming with a theory in which direct observation and experiment were impossible, developed the hypothetico-deductive method. Karl Popper worked on the hypothetico-deductive method in the twentieth century; its practice involves falsification of the hypothesis [52].

The scientific method is commonly said to have five main general steps:

1. **Making an observation -** Gathering and assimilating all the information about a process, event, etc. [53]
2. **Defining the problem -** Posing relevant and testable questions in the context of the existing knowledge [53].
3. **Formulation of a hypothesis -** Using inductive logic in order to consider the facts and information already known and formulate a tentative answer, explanation, or educated

guess, that is testable and falsiable, to the questions previously made [51], [53]. From the Popperian perspective, hypotheses are candidate explanations for observed patterns [54]. It is crucial to understand that a hypothesis has its roots in preexisting knowledge, which means that they will lead to results with a certain world-view built-in [55].

4. **Making predictions -** Applying a different form of logic - deductive logic – in order to make predictions based on the hypothesis. Deductive logic starts with a statement that is believed to be true, predicting which facts would also have to be true to be compatible with the initial statement [51], [55].
5. **Conduct the experiment -** Testing the hypothesis in a specific experiment/theory field [55]. After formulated, the hypotheses requires appropriate procedures to elucidate their answers without introducing personal bias. The questions always drive the procedures, for example, by exploring the predictions of multiple simultaneous hypotheses [56].

There are two general types of experiments, both of which compare data from different groups or samples. A controlled experiment manipulates factors being tested; comparative experiments compare non manipulated data from different sources, starting with the prediction that there will be a difference between groups based on the hypothesis [51].

The existence of a scientific method has been discussed over the years. Many scientists defend that a structured scientific method does not exist, while others claim that the scientific method is too simplistic [50]. Despite of the controversy, the scientific method is still very relevant, with its biggest strength relying on the fact that it is, theoretically, impartial: one does not have to believe a researcher or an experimental result – it is possible, in principle, to repeat the experiment and determine if certain results are, in fact, valid or not [55].

2.6 Chapter Conclusion

Through the previous topics, it was possible to properly understand which approaches are currently being used to perform GUI testing and what are the main characteristics of the different type of tools that support them. A wide range of GUI testing tools already exists and many new approaches and tools are constantly emerging.

Given that GUI testing tools can significantly vary from each other, even when applying the same testing technique, the selection of a tool should be wisely done according to the SUT characteristics, available time, and available funds before starting a GUI testing process. Selecting a less suitable tool could lead to a very annoying and less efficient testing process.

It is also stated that, from the perspective of these tools' developers, the effectiveness in defects detection should be evaluated before they achieve wide acceptance. A common way to assess test techniques is through empirical studies, in which different faulty software applications are experimented by them.

Taking into account the differences of the GUI testing approaches described above, and the absence of comparative studies that would allow us to draw more definitive conclusions regarding their behavior, it is raised the question of what are the differences concerning the effort required to perform activities prior to the test per se, as well as which is the most effective approach in terms of fault detection.

Therefrom, this dissertation work proposes to answer this question, comparing PBGT with the Random, Capture/Replay and Unit GUI testing approaches regarding their ability to detect errors/fault, their ease of use, and the efforts required to use them to test an application; regarding this, a description on how scientific experiments are performed as well mutation testing concepts were provided.

Chapter 3

The Experiment

In this chapter, we provide a detailed explanation of each of the steps performed in the scope of this dissertation study, from the formulation of questions to the practical work and the analysis of the obtained results.

First, the research questions are presented. The research questions served as a basis to establish a systematic methodology that would allow us to answer them. Therefore, an explanation of the metrics by which we attempted to answer these research questions is also provided.

Then, we provide an overview of the web-based applications we selected to test, and the GUI testing tools we chose to test the subjects are also addressed in terms of their capabilities and reasons that led to their choice. As previously mentioned, the tools are based on the testing techniques described in the "State-of-the-Art" chapter, and our case study's subjects are web-based and open source applications.

The practical work is then described. A general view on how mutations were applied to our subjects is provided. Then, we address the approach/methodology used to build the testing scripts and testing programs.

Finally, we present and discuss the obtained results of the tests performed.

3.1 Research Questions

The description of the different GUI testing approaches described in the second chapter established that the testing process with each of the techniques differs in many ways. Therefore, we conducted experiments on those approaches in order to determine the most feasible or appropriate to test each software application.

To accomplish this, we proposed the following research questions to answer to compare the random, capture/replay, and model-based (through the particular case of PBGT) GUI testing approaches:

RQ1) Which of the approaches is most effective in terms of failures detection?

RQ2) How much time does each approach require to test an application?

3.2 Metrics

After proposing research questions, we had to define the metrics to answer them.

Regarding the first question, we considered the "mutation score" metric suitable to evaluate the effectiveness of each approach — or tool — to detect faults. The mutation score value is given by the following formula, presented in [57]:

$$\boldsymbol{Mutation\ score} = \frac{\boldsymbol{Number\ of\ mutants\ killed}}{\boldsymbol{Total\ number\ of\ mutants}}$$

According to the formula, several mutants should be generated from each of the applications to be tested (the subjects of our experiment), and in order to compare the testing approaches, each mutant must be tested by the selected testing tools.

To answer the second question, we compared the time required both to perform tasks that precede the automatic tests, including understanding the tool's usage, creating test scripts, modeling the SUT or developing an own testing tool, and the time taken to perform the automatic test.

3.3 Subjects

To be eligible for our study, the software applications had to be open source, enabling us to mutate its files, and web-based. Moreover, the applications had to feature distinct use cases so that different usage scenarios could be covered. Additionally, the different ways of accessing various elements such as static elements, elements that only become visible by mouse-hovering over it, or dynamic elements needed to be represented by the applications since some of them can eventually influence the testing tool behavior.

Our set of subjects included the iAddressBook, TaskFreak and Tudu Lists web applications. In the following section, an overview of each is provided.

3.3.1 Subject 1 – iAddressBook

iAdressBook [58] is a PHP application through which it is possible to manage a list of contacts. It allows the user to add new contacts with different information associated, such as:

name, organization, affiliation, contact details, web page. In addition, and to keep the contacts list organized, different groups or categories of contacts can be created.

Thus, features as create, edit, delete and search contacts, and, for the categories, create, remove, add contact to or remove contact from are provided. It does not feature the login functionality since the user is meant to deploy the application in a local or remote web server.

iAddressBook "aims at 100% compatibility to the AddressBook in Mac OS" [58].

3.3.2 Subject 2 – TaskFreak

TaskFreak [59] is a simple web application that aims to efficiently manage projects and projects' tasks. As iAddressBook, it is written in PHP.

Projects, tasks and users can be created, edited and deleted. Different roles can be assigned to a user regarding his position in a certain project. Different ways to perform the same action are provided; as an example, tasks' percentage of completeness can be edited by using completeness bars to change it or a drop-drown list to choose the value. Moreover, it features different elements such as menus containing options only displayed by mouse-hover over them.

3.3.3 Subject 1 – Tudu Lists

Contrary to the the previously described applications, Tudu Lists [60] is a Spring/J2EE application. As the names implies, it is an application that aims to help its user managing lists of tasks (to-dos); however, in a different way than TaskFreak.

3.4 Subjects' test requirements

Below, we list the test requirements – described in a high-level view – that were considered for each one of the subjects of the experiment. Since our test requirements were defined by using a set of selected use cases, we can say that these use cases were used as a framework for defining our test cases.

3.4.1 iAddressBook

Category	Identifier	*Description*
Contact Management	TR1_1a	*Select contact*
	TR1_1b	*Add new contact*
	TR1_1c	*Edit contact*
	TR1_1d	*Delete contact*

	TR1_1e	*Delete several contacts (bulk delete on contact list)*
	TR1_1f	*Add to category*
	TR1_1g	*Delete from category*
Category Management	TR1_2a	*Create category "explicitly"/ "implicitly"*
	TR1_2b	*Delete category*
Filters	TR1_3a	*By contact's initials*
	TR1_3b	*By category*
Search	TR1_4a	*By contact's data*

Regarding the test requirement TR1_2a, we defined that a category is "explicitly" created when the task is completed by using the specific functionality provided to do so – the button "create category". By this, a category is created with its name defined by the string inserted in the input textbox reserved for this purpose.

In what concerns to the "implicit" creation, the definition we've applied refers to the case when a category is created when a new contact was added, as we explain next.

In the "new contact" form there is a field, named "category", that when not left blank makes the application verify if a category named equally to the passed string already exists in the database. If so, the application will only add a new contact to that category; on the other hand, if the category does not exist yet, it will be created and the contact will be added to it.

It can then be said that the implicit creation of a category also implies an implicit addition of a contact to a category.

3.4.2 Tudu Lists

Category	Identifier	*Description*
List Management	TR3_1a	*Add new list*
	TR3_1b	*Edit current list*
	TR3_1c	*Delete current list*
Todo Management	TR3_2a	*Add quick Todo*
	TR3_2b	*Add advanced Todo*
	TR3_2c	*Edit Todo*
	TR3_2d	*Delete Todo*
	TR3_2e	*Complete Todo*
	TR3_2f	*Show other Todos*

	TR3_2g	*Delete completed Todos*
Filters	TR3_3a	*Next 4 days*
	TR3_3b	*Assigned to me*
User Management	TR3_4a	*Login*
	TR3_4b	*Edit user data*
	TR3_4c	*Logout*

3.4.3 TaskFreak

Category	Identifier	*Description*	
User Management	TR2_1a	*Login*	
	TR2_1b	*Logout*	
	TR2_1c	*Access user data page*	
	TR2_1d	*Edit user data*	
	TR2_1e	*Delete user (admin only)*	
Project Management	TR2_2a	*See project*	
	TR2_2b	*Create new project (admin only)*	
	TR2_2c	*Edit project (admin only)*	
	TR2_2d	*Delete project (admin only)*	
Task Management	TR2_3a	*Add new task (by table & by Task menu)*	
	TR2_3b	*Edit task (by icon & by clicking table entry)*	
	TR2_3c	*Delete task*	
	TR2_3d	*Change task completion status*	
	TR2_3e1	*Comments*	*New comment*
	TR2_3e2		*Edit comment*
	TR2_3f	*Sort tasks*	
	TR2_3g1	*Filter*	*By user*
	TR2_3g2		*By context*

3.5 Testing Tools

In order to conveniently apply the considered GUI testing approaches to the selected subjects of our experiment, it was also necessary to choose and select tools that would represent each of these approaches.

Regarding the model-based approach, the choice will obviously be the PBGT tool, since it was the one that promoted the realization of this study, and it will be the approach which results will be the most relevant for us.

In what concerns the Capture/Replay approach, Sahi was the chosen tool.

Despite Selenium IDE being the more commonly used, as it is the most known Capture/Replay tool, Sahi might be a promising alternative when in comparison to it: it is a light and browser independent application, contrary to Selenium IDE that is implemented as a Firefox extension; it reports the tests' results in an html file, making it easy no evaluate them; it implements different DOM relation APIs that purport to facilitate the location of elements - when needed in specific cases - through easily understandable indications such as: _above; _under; _leftOf; _rightOf; etc..

Given these features, and given the fact that Selenium IDE is a tool we already know well, we have decided to use the Sahi application to perform our capture/replay tests, since that the time to prepare/configure and to learn the tools will also be required when considering the remaining approaches. In other words, knowing that the time required in activities prior to performing the tests per se will be considered, we wanted to have a similar level of knowledge of each of them.

Finally, there's the random testing approach. As already mentioned in the State-of-The-Art chapter, and besides the extensive search, is was not possible to find a tool that implements this approach to test web applications. The majority of the tools available for this type of test aim to test mobile and touchscreen applications, maybe due to the fact that the interaction with them is entirely based on "touches/clicks". Thus, we proposed to develop programs that would implement this approach, for each one of the subjects, representing our random testing tool. It was not possible to implement only one program capable of conveniently interact with any web application, for the reasons explained in the respective section.

3.6 Generating Mutants

For the reasons given in the section 2.5, "Mutation Testing", mutations need to be applied to the selected subjects.

To do so, certain files of the applications whose code was related to the functioning of the use cases we wanted to test were selected. Then the conditions of all the if-statements were changed so that the returned Boolean was exactly the opposite, i.e., if a certain condition was "true" it would then be "false." Therefore, mutation operators essentially replace some Boolean relations with their opposite.

For example, to mutate the following condition present in line number 233 of the iAddressBook's "actions.php" file:

```
if($contact == false) $contact = new person;
```

we modified it by replacing the operator "==" with "!=", resulting in

```
if($contact != false) $contact = new person;
```

Mutants outside the scope of this study were generated, and so these were discarded. Although the selected files have significant segments of code related to the considered use cases, some of the modified lines were related to others that were not considered; the mutants generated by modifying those lines were then discarded in the test phase. In addition, some of the mutations seeded only cause invisible faults, meaning that regardless of the action or sequence of actions, the behavior of the mutants is normal to the user's eyes and, by extension, also to the GUI testing tool. These were also discarded.

The Appendix contains tables with information regarding the considered mutants of each tested application. The mutated file as well as its modified line are identified along with the respective mutation applied.

3.7 The Procedure

At this point, the research questions were defined, the metrics through which we will evaluate the results of this experience were establish, and the study subjects and testing tools to use were chose. The mutations were also already applied to the study subjects and, thereby, the specific practical work inherent to our experiment took place.

As what had already been mentioned, each testing technique – with exception to the random approach - requires us to build a script or to craft and configure a model to properly test a software application. This means that diverse test cases have to be conveniently considered by the tester in order to cover the different use cases that were proposed to be tested.

Thus, in this section, we will explain the procedure adopted in order to build these elements. The first application for which the first tests were done was the iAdressBook, and it was the one that required the longest time due to the lack of knowledge on the use of the tools or technologies. For each one of the techniques, Random, Capture/Replay, Unit and PBGT GUI testing, we start by describing with more specificity the procedure for this first application, while for the remaining ones, only the details will be approached, since some of the procedures are the same or similar to others previously described.

3.7.1 Random Testing

Given that we could not find a tool to test web applications using a random testing approach, as previously mentioned, it was necessary to create a software program that would perform a random test for us.

Three programs were designed as "dumb monkey" testing tools, since they interact purely randomly without any information on how to proceed with the various SUT's elements. Selenium WebDriver was the API used to easily establish the interaction between the developed programs and the web applications. Both Selenium WebDriver and the study subjects are browser and operating system independent.

The following sections contain a description of the approach we've adopted to develop the programs. The complete code can be found in the Appendix.

3.7.1.1 IAddressBook Random Testing Tool Development

To begin, we started by analyzing the SUT's functioning and source code in order to identify and parse the elements with which the user can interact. Thus, there were parsed the <input>, <a>, <textarea>, <select> and <option> elements. The used API allows us to locate and select elements in different ways: by XPath, by tag name, by CSS selector, by class name, among others. In this case, the most suitable ones were the location *By.tag* and *By.xpath*, being the former used to select all the elements of a specific tag, and the later, the XPath selector, used in the case where there was a need to filter some of them.

While the usage of the *By.tag* selector is pretty much straightforward, the XPath selector evidently requires us to write an expression with information on what to locate. This method was used to find the <input> and <a> elements and, to clarify its usage, we briefly describe it in the following paragraphs.

Regarding the <input> elements, we didn't want the application to interact with those having the argument *type* equal to 'file' or 'hidden', as they are related to use cases of files manipulation (which are not covered by our tests), or they are not shown to the user (making it not possible to interact with them), respectively.

Thereby, the desired elements were collected and stored in a list, using an XPath expression to locate them, as follows:

```
List<WebElement> allInputElements =
driver.findElements(By.xpath("//input[@type!='hidden' and @type!='file']"));
```

Furthermore, the application's elements corresponding to anchors (<a>) that were not related to the considered use cases, or that redirect to pages outside the application, weren't also intended to be selected and, once again, XPath was useful to filter them. The required <a> elements were then stored in another list:

```
List<WebElement> allLinkElements = driver.findElements(By.xpath(
       "//a[@href!='http://iaddressbook.org/' and @href[not(contains(.,
'export_vcard_cat')) and not(contains(., 'export_csv_cat')) and
not(contains(., 'export_ldif_cat')) and not(contains(.,'import_folder'))]]"));
```

Having the elements parsing task been completed, we wanted the application to randomly select an element to interact with. To do so, a new list – *allElements* – consisting of a union of all the created lists, i.e., a list containing all the collected elements, were created. From that list, one element is then randomly selected and, according to its tag and respective arguments, an interaction is applied, which can be a "click()" or "sendKeys()" (with a sequence of random characters too) action.

If the selected element's tag is equal to <a>, <select>, <option> or <input> with the argument *type* different from "search" and "text", a click in element instruction is executed. In the remaining cases, namely the cases where the element's tag matches <textarea> or <input> with the argument *type* equal to "search" or "text" (input type='search' or type='text'), the sendKeys() instruction is the one applied.

There was no need to consider different type of interactions since only these two are needed to use iAddressBook.

The process is then cyclically repeated after each interaction. Thus, all the elements available in the new state of the application are collected, one is randomly selected, and an interaction is applied. To make the cycle - and consequently our application - stop looping and running after a certain amount of time during which no error was found, we introduced a variable to conveniently define it. In our tests, we've set that variable to 1800000 milliseconds – 30 minutes.

It should be noted that, after selecting an element, the "element.isDisplayed( )" condition is evaluated before applying an interaction. This will verify that the element is visible in the application's current state, being it then possible to interact with it.

Although it wasn't verified in iAddressBook, in some other tested applications this condition was not sufficient to ensure that all the elements were visible in any application's execution state. As an example, the opening of a non-modal window can overlap some elements, being the "isDisplayed( )" method unable to verify that they aren't available to the user at that moment. To cover such cases, the instructions "click()" and "sendKeys()" should be surrounded by a try-catch statement to handle the inability to correctly execute them.

Another situation we had to deal with is the opening of modal boxes/windows.

WebDriver requires information of when they are present to change the interaction context to them or, otherwise, it will throw an “UnhandledAlertException: Unexpected modal dialog” error.

The solution here passes by inserting some instructions to wait for and switch to the alert (when it is present) after executing an interaction – the WebDriverWait class can be instantiated with a waiting timeout set to at minimum 1 second.

```
WebDriverWait wait = new WebDriverWait(driver, 1);
wait.until(ExpectedConditions.alertIsPresent());
Alert alert = driver.switchTo().alert();
```

However, this approach has a significant impact on our program’s performance since it approximately triples the amount of time necessary to execute the same number of interactions. Therefore, and taking into account the reduced dimension of the subset of elements for which an interaction shows a modal window, it was decided to add the abovementioned statements only for these elements.

Hence we began to identify all the elements that trigger those windows. Some were only detected by running our developed program and waiting for the “Unexpected modal dialog" error, as it is not easy to find them all manually. It is necessary to inspect the SUT’s source code so that we know how we could locate a specific element with the WebDriver supported selectors.

Then, and in order to only apply the “wait for an alert” instruction after an interaction with one of these elements, we’ve added conditions similar to the following one to identify them:

```
if (element.getAttribute("value").contains("delcon") ||
    element.getAttribute("value").contains("catdel"))
```

In this particular case, the condition is checked when the element is an <option> (i.e., its tag name equals “option”). Thus, if this condition is verified it will execute the desired portion of code, which contains the action to be performed and the respective wait for an alert, since the selected element is known to trigger a modal dialog.

It should be noted that the introduced wait for and switch to the alert instructions were surrounded by a try-catch statement to handle the situation where, for some reason - mostly due to the introduced mutations - the modal window did not open.

To easily identify the sequences of actions that lead to an error in the iAddressBook’s execution, each selected element and performed action were printed on the console, which can be seen as a log registry of all the interactions.

3.7.1.2 TaskFreak Random Testing Tool Development

Having the foregoing description addressed the approach used to implement the program with detailed explanations, in this section only specific cases will be detailed, as referred.

Contrarily to the iAddressBook, the TaskFreak application features the login functionality. Since the random application would hardly be capable of logging in to test the rest of the application, a sequence of instructions was introduced in the program so that the login could successfully be done when on that page.

Thus, the following condition was introduced in order to be checked at the beginning of each of the previously explained cycles.

```
if (driver.getCurrentUrl().contains("login.php")) {
WebElement element1 =
driver.findElement(By.xpath("//input[@name='username']"));
Element1.sendKeys("admin");

WebElement element2 =
driver.findElement(By.xpath("//input[@name='password']"));
element2.sendKeys("");

WebElement element3 = driver.findElement(By.xpath("//input[@name='login']"));
element3.click();}
```

The code will always check the URL of the application between each interaction, and if it contains the string “login.php" (meaning that it is on the login page), it will locate each of the required elements and will conveniently interact with these in order to provide information the login information. In this particular case, the sequence of interactions is performed in an orderly manner.

Then, we proceeded the same way as described in the previous section. That means that the elements of the application were collected, with this one having one more type of elements with which it is possible to interact to, the <*th*> elements that allow a user to sort the table’s content.

It is important to note that, in this application, the opened windows are not modal, meaning that the application can interact with the remaining elements of the page. This particularity made the application to sometimes try to interact with elements that were behind of one of these opened windows. The *isDisplayed()* function does not cover this situation, since the selected element is considered to be displayed. The elements that were susceptible for this to happen were the <*a*> and <*th*> ones, so the following code was added in what concerns the interaction with them.

```
try {
        element.click();
} catch (Exception e) {
        System.out.println("Couldn't click on element: covered by another one");
}
```

As already mentioned, the *isDisplayed()* function will return “true” if the seleted element is displayed, and “false” when the opposite happens. Something that was not also possible in the iAddressBook application is the possibility to interact with some elements for which the *isDisplayed()* function returns “false”. In TaskFreak, these are elements that only become visible by mouse-hovering over them. If the interaction with these <a> elements was implemented in a similarly way as what was described – using the “element.click()” function - it would not evidently be possible.

With that said, and knowing that the hidden elements were already filtered by the XPath expression, it is possible to conclude that, when the *isDisplayed()* method returns “false”, the element the program selected is one of these elements that are only visible by mouse-hovering over them. Thereby, we use a JavaScript function to make it possible to deal with these specific elements.

```
JavascriptExecutor executor = (JavascriptExecutor) driver;
executor.executeScript("arguments[0].click();", element);
```

The remaining cases exposed by the application were already detailed in the scope of the explanation of the previous random testing program’s development.

3.7.1.3 Tudu Lists Random Testing Tool Development

The third application didn’t present any scenarios that weren’t previously explored. The only difference is that, in addition to the <th> interactable elements – which were already mentioned in the previous section - this one also introduces <td> elements.

With the help of an XPath expression, we defined that only the clickable <th> and <td> elements should be selected to interact with, i.e., the <th> elements which argument “class” contains the string “sort”, and the <td> elements that contain the argument “onclick”.

```
List<WebElement> allSortableElements =
driver.findElements(By.xpath("//th[@class[contains(., 'sort')]]"));

List<WebElement> allClickableTDElements =
driver.findElements(By.xpath("//td[@onclick]"));
```

The time this third testing program - to test the Tudu Lists application - required us to be implemented can be summarized as the time it took us to study the SUT in order to: conveniently select the interactable elements; verify the opening of modal and not-modal windows in the different elements; verify the existence of interactable and non-displayed elements.

Given that we had the previous programs code, everything went much faster.

3.7.2 Capture/Replay Approach

To perform the tests with a capture/replay GUI testing approach, the Sahi testing tool was the one we chose.

Sahi records the sequence of actions in a script with a language of its own, which features a range of functions. Different types of assertions can be inserted, and different forms to easily locate elements can be used – which are especially useful in cases where the script has to be manually edited, since that sometimes the default methodology is not the most suitable, as will be described below. Besides being enriched with different functions, it also allows us to (manually) insert JavaScript code in the script, if really needed.

The details of use, mainly the ones to which the manual edition of the script was required, will be approached throughout the description below.

Despite being available a free and open source version of the tool, we've used a trial version of the professional one, since it features an integrated environment to manually edit scripts. Regarding the recording of a script, there are no differences between both the free and the paid version.

Once again, we refer that after a more detailed description of the approach adopted to build the test suite to test the first application – iAddressBook - is provided, the following ones will only address situations that required more attention to some specific details.

3.7.2.1 Concepts

Throughout the next sections, some specific concepts as well as some designations proposed by us will be used. Here we describe them.

Assertions are instructions used for “"verification" or "checking" of functionality” since they “compare runtime behavior against expected behavior” [61]. A script should contain them after specific sequences of actions so result of the different test cases could be verified.

We will distinguish assertions introduced “automatically” and assertions introduced “manually”. The former refers to assertions generated by the tool, introduced by mouse-hovering over the element to which visibility and content we intend to verify, and pressing the *Alt-key* to generate assertion code. That will consist of a set of assertions to be applied regarding the selected element, namely: *_assertExists(); _assert(_isVisible()); _assertEqual(); _assertContainsText().*

The former, assertions inserted manually, are generally those that aim to verify that an element is not present – so we cannot mouse-hover over it to automatically generate the code.

3.7.2.2 Building the iAddressBook test script

In this section, we will describe the process – implying the recording and manual edition – involved to build the test scripts according to this testing approach. Thus, and through a high-level perspective, we describe the approach adopted to test each of the considered use cases. When applicable, there's a point named "detail" that will describe certain situations that slowed down the process, requiring us to manual edit the test script.

Test Requirement: Add new contact
Description: First, we accessed the "new user" page and filled in all the fields presented there, excepting the middlename - it will be filled in later when editing the contact.

After the contact was saved, we've verified that the entry was really there by clicking in the first letter of that contact's lastname (the way the application lists the contacts) - which is, in this case, the letter 'G' - and an automatic assertion was inserted – verifying the existence, visibility and equality of the test. The assertion code was generated as follows:

```
      _assertExists(_link("Gyllenhaal, Jacob"));
      _assert(_isVisible(_link("Gyllenhaal, Jacob")));
      _assertEqual("Gyllenhaal, Jacob", _getText(_link("Gyllenhaal, Jacob")));
      _assertContainsText("Gyllenhaal, Jacob", _link("Gyllenhaal, Jacob"));
```

Test Requirement: Edit contact
Description: First, the profile of the previously added contact was accessed. To verify that the page was opened, automatic assertions were inserted to verify the existence of the *div* in the top of the page, which content corresponds to the "title firstname middlename lastname" of the contact.

In that page, we clicked the "edit" button to access the contact editing form. The field that previously was left blank (middlename) was filled-in, and the content of the field "lastname" was edited. After saving the modifications, a new assertion related to the text of the profile's top *div* was inserted so we can verify that the contact was correctly edited.

Test Requirement: Delete contact by checkbox
Description: We've clicked the first letter of the contact's lastname, so it was listed, and "checked" the checkbox that corresponds to that element. Then, we clicked the "delete contact(s)" link, which opens a modal window for us to confirm the contact deletion. The lines that verify that the window did open are automatically inserted by the tool, having the following ones been appended to the script:

```
    _expectConfirm("Do you want to delete the selected contacts?", true);
    _click(_link("delete contact(s)"));
    _assertEqual("Do you want to delete the selected contacts?",
_lastConfirm());
```

After the contact was deleted, we've inserted an assertion manually to verify that the contact is, in fact, no longer listed:

```
    _assertNotExists(_link("Contact, Delete"));
    _assertNotVisible(_link("Contact, Delete"));
```

Detail: The id of the checkbox to select a contact is not dynamic, but dependent on the element's index in the database. Given that, we've used the leftOf("element") Sahi's function in to "check" the checkbox corresponding to the contact we want to select.

```
      _check(_checkbox(0, _leftOf(_link("Contact, Delete"))));
```

If the tables of the MySQL database were internally cleaned between each test, the indexes would always be the same. However, we've opted to identify it this way since it is more specific.

Test Requirement: Delete contact by button
Description: To design this test, we've accessed the contact's profile page, where the "Delete Contact" button can be found, and clicked it.
Then, an assertion was manually inserted to verify that the contact is no longer listed.

Test Requirement: Search contact
Description: To test this use case, 3 test cases were considered: search for a non-existing name; search for the firstname of the contact; search for half of the lastname of the contact.

To test the first case, i.e., to verify that no contact is listed when using search terms that are not associated with any of the existing contacts, we've filled-in the searchbox with the string "non-existent ". Since the search should not return any results, an assertion was inserted automatically to verify the presence of the "no contacts" text that is showed in this case.

Regarding the second case, a string consisting on the firstname of the contact to be searched was used, and it was verified that the profile's page of the contact which name is the most similar to that string is opened

The following lines were appended to the script to verify the existence of this text in the top of the profile's page.

```
    _assertExists(_div("Mr. Test Goodtest"));
    _assert(_isVisible(_div("Mr. Test Goodtest")));
    _assertEqual("Mr. Test Goodtest", _getText(_div("Mr. Test Goodtest")));
    _assertContainsText("Mr. Test Goodtest", _div("Mr. Test Goodtest"));
```

Finally, we've search for the contact using half of the string of his lastname. The application must return the same element, since no other contacts containing the string "good" exist in the database. The assertions were added the same way as previously.

Detail: After filling in the searchbox, the iAdressBook requires the "Enter" key to be pressed so it can to proceed with the search. It was necessary to manually edit the script to add an instruction to simulate that behavior. The instruction is the following:

```
typeKeyCodeNative(java.awt.event.KeyEvent.VK_ENTER)
```

Test Requirement: Create category (implicitly and explicitly)

Description: Regarding this test requirement, there are two test cases to be considered: the explicit and implicit creation categories, as explained in the "Test requirements" section.

Thus, we started by adding a new contact with the field "category" filled-in – so a category could be implicitly created. Then, another category was explicitly created.

The categories are listed in a drop-down list and so it not possible to introduce automatically insert assertions for this type of elements. However, we didn't think it was relevant to manually introduce one, as the testing tool would acknowledge a fault when trying to select an inexistent category in the following tests - being indirectly detected, then.

Test Requirement: Add contacts to categories

Description: Having categories been created, it is also intended to test if contacts can be correctly added to them.

First, one contact was added to the category previously created in an implicit way. To do so, the contact was selected using the leftOf(element) function, for the reasons previously explained, and the option "add to 'category'" was chosen using the drop-down list.

Then, a contact was added to the category explicitly created. As a result, we will have two contacts in the implicitly created category (one of them previously added in the implicit creation of category), and other in the explicitly created one. To verify that these contacts are actually in the category they should belong to, assertions were automatically inserted; for example, the following assertions verify if contacts belonging to the implicitly created category are in fact listed:

```
      _assertExists(_link("One, Test"));
      _assertVisible(_link("One, Test"));
      _assertEqual("One, Test", _getText(_link("One, Test")));
      _assertContainsText("One, Test", _link("One, Test"));

      _assertExists(_link("Three, Test"));
      _assertVisible(_link("Three, Test"));
      _assertEqual("Three, Test", _getText(_link("Three, Test")));
```

```
        _assertContainsText("Three, Test", _link("Three, Test"));
```

Detail: As it can be seen in the image below, the application iAddressBook contains two same-named elements ("Test 1") in a drop-down list.

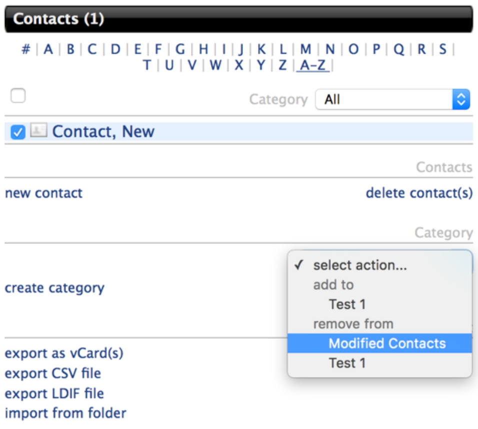

Figure 2: iAddressBook – Specific Case

If we wanted to remove a contact from the category, we would naturally have to select the second "Test 1" element (the one bellow "remove from").

This is the code generated for this action:

```
    _setSelected(_select("cat_menu"), "Test 1");
```

However, we verified that the code generated when adding a contact to this list – corresponding to the first option in the image – is exactly the same.

This happens because, by default, the capture/replay testing approach identifies an element using the text by which the element is presented to the user, and not its internal identification. Thus, in the "replay" phase, the tool will always select the first option with this name, independently of which one we wanted, since it's the first element matching the selection.

If desired, it would be necessary to manually add an XPath expression to correctly identify the element, as follows:

```
_setSelected(_select("cat_menu"), _byXPath("//option[contains(@value,
'delcon') and text()='Test 1']"));
```

Test Requirement: Delete a category

Description: In order to test the category deletion, the category "Test" was selected in the drop-down list of categories and, in the drop-down list of options to apply, the option "delete category Test" was selected.

The following assertion was manually inserted to verify that the category was removed – it verifies that the option "Test" (name of the category) does not exist in the categories list.

```
    _assertNotExists(_option("Test", _in(_select("cat_id"))));
    _assertNotVisible(_option("Test", _in(_select("cat_id"))));
```

3.7.2.3 Building the Tasfreak Test Script

The approach adopted to build the test scripts for this application was similar to the previous one. To test functions of adding and editing users, projects and tasks, many test cases were considered - leaving each of the compulsory fields empty and using repeated usernames. Automatically created assertions were inserted to verify the existence of the error messages showed in each case.

After editing the elements filling-in all the fields correctly, assertions to verify if the details have really changed were inserted. Also, when a certain element is deleted, assertions were inserted manually to verify that will definitely not listed.

The different available buttons to perform the same action (add, edit, delete) were tested. These could be real buttons, images or textual links.

Regarding the tests of filters, each of the defined in the tests requirements was selected and assertions were inserted so it could verifiy that the elements that should be listed are actually listed; the inverse was also done, meaning that manual assertions were added to verify that the elements that shouldn't be listed are not listed in fact.

The following points describe cases that were not addressed regarding the previously built script (to test the iAddressBook application).

Test Requirement: Login and logout
Description: Several and different cases - such as inputting a non-existent username or a wrong password - as well as the case of successful login were considered.

The existence of an error message was verified to each of the test cases that lead to an invalid login. After correctly login in, the existence of the logout button was asserted.

Regarding the logout, an assertion was inserted to verify that the logout message is showed right after the click on the button.

```
    _click(_image("frk-logout"));
    _assertExists(_cell("You are now logged out. Goodbye."));
    _assertVisible(_cell("You are now logged out. Goodbye."));
    _assertEqual("You are now logged out. Goodbye.", _getText(_cell("You are
now logged out. Goodbye.")));
    _assertContainsText("You are now logged out. Goodbye.", _cell("You are now
logged out. Goodbye."));
```

Test Requirement: Edit task completeness
Description: In one of the cases of editing tasks, the percentage of completeness of a certain task can be changed using images (the completeness bars), so it was necessary to access the task's description to verify, textually, that the percentage was changed.

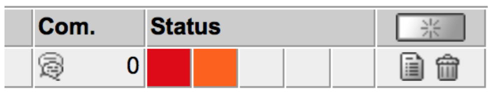

Figure 3: TaskFreak – Completeness Bars

Test Requirement: Sort
Description: To verify that the elements are sorted after clicking on the button available to sort them by task and by user's name, the approach must be slightly different.

After searching, it was found that Sahi allows us to identify a table's cell from its row and column.

With that said, an initial assertion was manually inserted to verify an element is on a specific position in the table. After clicking the sort button, a new assertion was inserted with the updated and supposed position.

The process was performed for both buttons (sort by name and sort by task).

Following, we present the lines that test the sort by task function.

```
      _assertEqual("Test", _getText(_cell(_table("taskSheet"), 5, 4)))
      _click(_tableHeader("Project"));
      _assertEqual("Test", _getText(_cell(_table("taskSheet"), 1, 4)))
```

3.7.2.4 Building the Tudu Lists Test Script

Finally, no new techniques were required to build the script to test the Tudu Lists application.

The only small difference is that this application shows the percentage of completion of a certain to-do list, calculated by the ratio of completed and to be completed tasks. Thus, assertions have been inserted to verify that this value changes correctly when a task (to-do) on the list is completed or when new tasks are added.

All the remaining test requirements have been treated similarly to those previously described. Implementation of actions such as adding, editing or deleting elements, and selecting filters, as well as verifying the correct result of those actions, was conducted in accordance with the above approach to build the script to test each application.

The complete generated scripts can be found in the Appendix.

3.7.3 Unit Testing

As mentioned in the second chapter, the creation of unit tests is a time consuming process and these tests would hardly find errors that are not detected by the Capture/Replay testing approach. Capture/Replay testing tools usually provide a feature by which a Java code for unit tests can be automatically derived from the scripts generated. Since writing the code all manually would not improve error detection, we have chosen to generate it automatically, so we can evaluate the resultant file regarding the number of lines of code (LOC).

This approach was adopted to help us to infer an approximate time needed to create the test files manually, although a study [62] states that LOC metric used individually is not a reliable indicator of effort. It is suggested by Shihab et al. that a better assessment can be obtained by combining the metrics complexity, size (LOC) and churn; however, we decided to rely solely on the LOC value due to the impossibility of evaluating the remaining two.

A study [63] states that on average a programmer spends one hour to write ~22 lines of code in Java, as depicted in the graph below, taken from that article (Figure 2),. Based on this calculation, we calculated the effort, measured in average time, required to apply the unit testing approach.

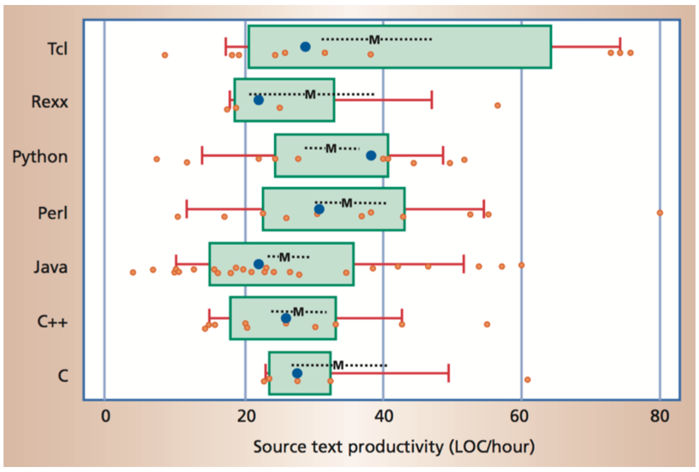

Figure 4: Productivity in lines of code per work hour. Graph from L. Prechelt, “An Empirical Comparison of Seven Programming Languages,” *Computer (Long. Beach. Calif).*, vol. 33, no. 10, pp. 23–29, 2000.

3.7.4 Pattern Based GUI Testing

Regarding the tests performed to assess this approach, we have used the models and results that were obtained from other experiments conducted by a team of researchers within the scope of the PBGT project. These models were built based on the same test requirements.

When configuring them, and contrarly to what happens in the Capture/Replay approach, there are more "artifacts" that replace the assertions. For example, to represent the interaction with a button or a link, a "**call**" UI test pattern should be added to the model. To evaluate the result of the interaction with such elements, there is a set of different checks that can be selected and defined: ChangePage and StayOnSamePage, which verify that the URL of the SUT have either altered or remained the same, respectively; PresentInPage and NotPresentInPage that represent the presence or absence of a certain text in the page presented after interaction. This way, information related to the expected behavior after each interaction can be provided.

The "**input**" patterns represent fields through which we fill in information – a string – to be submitted. These can be valid or invalid and, regarding the latter, an error message associated to an invalid input can be defined.

The "**login**" UI test pattern can be defined as a mix of the two previously referred patterns; it contains two input elements - corresponding to the username and password fields - and call corresponding to the submission button. Different test cases, can be then defined.

It is important to note that the sequences generated by the PBGT do not follow a linear order. Thus, when generating the test script, different paths to transverse the model are generated, allowing different sequences of actions to be tested.

PGBT provides different strategies by which the script can be generated; however, we have only considered the results the research team obtained when applying the random strategy, since it was shown to be the most effective in detecting failures.

3.8 Results

First, it should be mentioned that the obtained results could differ slightly if the created programs, test scripts, and application's models were built and crafted by others, as these are dependent on the tester's perception of the tested software.

After creating the necessary artifacts, each mutant was tested by the selected testing tools. For each of the testing approaches, we've annotated the time it took to create test scripts, or programs, in the case of the random testing strategy, as well as the average time taken to kill a mutant, i.e., detect a fault in a mutant.

Moreover, we collected the results of each test performed on the mutants of the subjects with each of the testing approaches. Appendix contains the complete tables of results.

3.8.1 Time Required to Build Tests and Find Errors

The results regarding the time required to build the tests, as well as the time each tool takes to kill a mutant, are presented below in two subsections. These results help us to answer our first research question. It must be mentioned again that the results regarding the efforts required by the Unit testing approach are solely based on an estimative, as previously described in section 3.7.3. These were calculated by establishing a relation between the size of a file, in terms of lines of code (LOC), with an estimated time taken to write them.

3.8.1.1 Time Required to Build Scripts and Programs

Table 1: Efforts (in time) required to build the programs/scripts and craft and configure models.

GUI Testing Approach	**Subject**			**Average Time**
	iAddressBook	**TaskFreak**	**Tudu Lists**	
Random	7h:20min	4h:15min	1h:50min	4h:28min
Capture/Replay	3h:50min	2h:55min	1h:10min	2h:38min
PBGT	2h:17min	2h:44min	1h:13min	2h:05min
Unit (estimated)	12h:25min (LOC = 273)	-	-	-

First, we must say that only the scripts built to test the iAddressBook application based on the Capture/Replay approach were converted to unit tests. Given that the results were completely elucidative, we did not think it would be important to consider the remaining ones since we know the value would always be high; thus, only the three other approaches are addressed in the following paragraphs.

The first random testing program built - to test the iAddressBook application - took some time to be implemented, as it is possible to observe. We did not have the knowledge on how to use the Selenium WebDriver API, and we chose to study the API and develop the random testing program simultaneously. As a result, the initially conceived approach for the construction of the program had to be adapted to function with the WebDriver's characteristics discovered during the process. After developing the program, several iterative corrections were needed in order to eliminate some errors found after running it. Thus, the time required to build this first program was considerably higher. Thus, we can say that time to build the first test suite to test iAddressBook, with respect to the random testing program, is excessive since it includes the time spent to learn how to use the tool and the time to build the script itself, given that both tasks were performed simultaneously.

The second random program also required special attention to some details, previously described in the respective section, which led to some setbacks during development. That is, the

TaskFreak structure presented new scenarios, requiring us to spend some additional time on searching for solutions to implement the test program properly.

The last program developed took considerably less time, since we already had the necessary knowledge to handle different situations with WebDriver.

Regarding the use of the Sahi tool, for the implementation of tests based on the Capture/Replay testing approach, the exploring-to-understand its functioning had also taken some time since it is not a very intuitive tool. Moreover, it is not a popular application compared to other tools such as Selenium IDE, so we spent a considerable amount of time looking for information to deal with certain situations already described (e.g. the need to use different types of selectors and the need to manually insert certain instructions).

Regarding the PBGT approach, the collected times refer to the time spent by a team of researchers who developed the models in the scope of the PBGT's researching project. The team already had the knowledge needed to use the application, so no time was spent to learn it.

Overviewing the results, we can we can say with certainty that the more time-consuming strategy regarding the activities prior to performing the tests was the random strategy. Even if the first value – regarding the iAddressBook application - is influenced by the inclusion of learning time, the short value of the time spent to developed the Tudu Lists random testing program is not also truly representative because we already had the previously developed programs as a base; in what concerns Capture/Replay and PBGT testing approaches, and discounting a few minutes to the time spent on creating the first test suite with the Capture/Replay technique, the time taken by both was very similar.

3.8.1.2 Time Taken to Kill Mutants

Table 2: Average of time taken by each approach to kill a mutant.

Testing Approach	**Subject**			**Average Time**
	iAddressBook	**TaskFreak**	**Tudu Lists**	
Random	4min	1min	1min	2min
Capture/Replay	50sec	1min	1min:20sec	1min:03sec
PBGT	41min:12sec	51min:12sec	26min:30sec	39min:38sec

As said, once programs, scripts, and models to generate test scripts to test the mutants were built, each mutant was tested with each testing tool. The table above presents the average time each approach took to kill a mutant. We have only considered the time in cases where failures were actually detected, discarding the cases where all the tests passed. These passing cases were discarded because for two main reasons: random testing program would run until the time we've defined is over, making no sense to consider it; PBGT can also takes some time to run the entire test script since it was generated accordingly to the random paths strategy. As a result, many test cases are tested and the resultant value of elapsed time would significantly skew the metric we're studying, average time to kill a mutant.

As shown in table 2, the Capture/Replay is the approach that detected faults more quickly. We can then say that besides being one of the approaches that allowed us to build the test script faster, is also the one that detected faults more quickly. Regarding this metric, the fact that it is a fairly simple approach, consisting basically in accurately replay/reproduce a sequence of defined actions and evaluate assertions' results, can explain the results.

3.8.2 Tests Results

This section presents the tests results regarding faults detection/killed mutants by each of the testing approaches. It is divided into four subsections: the first one addresses the overall results in terms of the "mutation score" metric, presented previously in the "Metrics" section; the remaining three sections provide a detailed explanation of the cases when only one of the testing approaches was able to kill a mutant, regarding each of the three subjects, which we have denominated "particular cases".

Complete tables of tests results are in Appendix.

3.8.2.1 Mutation Scores

Table 3: Mutation scores for each subject.

Testing Approach	**Subject**			**Overall Mutation Score Value**
	iAddressBook	**TaskFreak**	**Tudu Lists**	
Random	4/68	9/77	1/28	8,09%
Capture/Replay	55/68	66/77	22/28	82,66%
PBGT	62/68	67/77	26/28	89,6%

As it is possible to see, the PBGT is the approach that scored a higher value of mutants killed (89,6%). The Capture/Replay shows a little lower value (82,66%), while the Random approach fell far short on detecting failures since the mutation score value achieved with this technique is significantly low (8,09%).

3.8.2.2 iAddressBook Particular Cases

Mutants 32 and 34

The mutation introduced in mutants 32 and 34 causes the contacts' filters "#" and "Z", respectively, to fail on listing the contacts whose last name starts by a numeric character or by the letter "Z".

Since the random testing approach does not have any information about the application's expected behavior, it was already anticipated not to kill the mutant.

The Capture/Replay approach was not able to detect the fault because when building the test cases to test search filters, we had not tested each of the existing characters (or filters). If we wanted to verify that each of the letters would actually only list the contacts whose last name starts by that letter, we would have to add a contact to each of them, click on each one of them, record the interaction on the script, and manually insert assertions to verify that each contact is only listed when filtered by the letter that corresponds to it and not by the remaining ones.

In our script recording session, we only specified one case, checking that the contact was only listed by the letter corresponding to it and not by another selected one.

PBGT killed the mutant. Since it is easier to model the test requirement, all the filter elements were added to the model which, therefore, was able to generate a script that detected the fault.

Mutant 41

This mutant does not add the selected contact to category (regarding explicitly created categories). In fact, other contact is added instead of the selected one.

The Capture/Replay approach detected the fault because after a certain contact is added to one category, the script selects that category and one assertion to verify that the selected contact is there is evaluated.

With the PBGT approach, the sequence of actions executed was not linear, so the tool could not determine which contact was already added. Therefore, the tool only checked that the category was not empty. Since a wrong contact was there, it passed the test. By that, we must say that if a sequence of actions was defined in the model in order to check if a certain contact is in the category right after being added to it, PBGT would also detect the failure.

Mutants 67, 88, 89, 98, 104 and 110

For these mutants, an error on saving contact/category showed although it did not occur, as indicated.

PBGT was the only tool able to detect the failure. The validity of the input fields was set to "valid", and after submitting the information, an error was reported even though the information was correctly stored in the database.

In the case of Capture/Replay approach, no assertionNotVisible was inserted to verify that this unknown error message did not show. Thus, the fault was not detected because the applications' functioning was completely normal.

Mutant 90

Mutant 90 makes the application fail to show the category that a contact belongs to in its profile page.

PBGT was the only one that killed the mutant because its generated test script tried to select the category from the contact's profile page. Since the category was not there, the failure was detected.

The Capture/Replay scripts built did not attempt to access a category that way and thus the mutant was not killed by those scripts. Another way to detect the fault would be inserting assertions for each of the visible elements, but that would be a time-consuming task especially in more complex applications.

Mutant 91

Mutant 91 had two visible faults.

The first one caused the contact's category not to be listed in its profile, as described above. The second fault resulted in a false error message, "DB error on find" was shown after saving a contact.

For the reasons mentioned above, only PBGT was able to detect both faults.

Mutants 185 and 187

Mutants 185 and 187 do not allow a contact's prefix and middle name to be saved, respectively. These mutants were killed by the Capture/Replay testing approach but not by PBGT.

Capture/Replay detected faults because when a particular contact's profile page was opened, an assertion was evaluated in order to verify that a *div* containing a text that corresponds to the full name of that contact was showed, so the tool could verify that the page opened. Since that *div*'s text was not the expected, the error was reported.

Since PBGT has other mechanisms to verify that an application opens a page, the check "presentInPage" was not introduced. Besides adding that specific check to detect the failure, it also could have also possibly been detected if the validity of the "input" pattern corresponding to these fields was set to "valid" - depending on the SUT's behavior, PBGT could eventually detect that "valid" inputs were not correctly handled.

3.8.2.3 TaskFreak Particular Cases

Mutant 12

This mutant allows a user to be added to a project he does not belong to. In other words, in TaskFreak a task needs to be assigned to projects and users; thus, while adding a new task, when the field "project" is filled-in before filling the field "user", the users that would be listed in the "users" drop-down list are the ones belonging to that project. When done in reverse, which means when the "user" is selected before the "project", the application is expected to automatically change the user to a default one if the selected does not belong to the project. That does not happen in this mutant.

Since our capture-replay script fills the fields in a regular order, this mutant was not killed,

Regarding the PBGT approach, these fields are marked as "any order" in the PARADIGM (PBGT) model, since they are inside a "form" UI Test Pattern and not linked by any sequence; thus, several test cases in which the user field is altered first than the project field were generated.

Mutant 40

The faulty mutant 40 does not allow changes to a given task's completion percentage by "moving" the completeness bars. In PARADIGM model, the click on these bars is, of course, represented by a "call" element. This call is inside a "group" element that, in turn, contains a set of other elements related to the task edition form and with which PBGT interacts in a random sequence, depending on the path generated.

Other than changing their color, interacting with the completeness bars does not make the page's URL or content change. Therefore, the verification that the percentage had definitely changed would be done by analyzing the percentage in textual mode which is present in the task's details page. To do so, that page would have to be refreshed in order to check the new value. It could be thought that, given the fact that PBGT integrates Sikuli, some "checks" to images could be used; however, Sikuli is only used to map elements and locate elements - via image recognition - when Selenium can not locate them.

PBGT verification would have to be modeled by a specific sequence of actions, since the sequence of interactions with non-linked elements is randomly generated, to kill the mutant. The Capture/Replay testing strategy detects the failure precisely for this reason; the sequence of actions in the script is linear and an assertion was inserted in the respective page right after clicking on the bars.

Mutants 51 and 92

Mutant 51 makes the red box that contains a message informing of errors in the fields related to the username/password not appear on the login page.

This fault was only detected by the Capture/Replay approach. The test scripts were developed to verify the occurrence of the interaction with the application after submitting wrong login data, and did not exactly test the presence of the red box. In this approach, these errors were detected by test cases designed to test the "login" use case.

In the PBGT testing approach, different checks and validation methods were used. The "validity" of invalid test cases—submitting a wrong username/password—were naturally defined as "invalid" and, in relation to the "Submit" button of the login UI test pattern, we selected the "StayOnSamePage" check for this case.

PBGT allows us to set the error message that should appear when a field is defined as "invalid," however, none had been introduced in this case; if a message was defined, PBGT would have killed the mutant.

The mutant's 92 error is similar, although the red box displays when an unnamed project is created. In PBGT, we configured this test case (considering the empty string one of the inputs to the project's name field) as invalid, and the message associated to that invalidity was "This field is compulsory." We had not selected/defined a "presentInPage" check to verify that the text, "There are some errors in the form - Information not saved," in the red box appears at the top of the original application's page.

In mutant 56, for example, such an error has occurred in another page of the application and none of the approaches detected the fault.

Mutant 93

In mutant 93, a tag text is not shown in the login page. Since the presentInPage check containing the string "Fields in red are compulsory" was defined, PBGT was able to kill the mutant.

However, our Capture/Replay script did not detect it due to the lack of assertions inserted in order to verify the existence of that tag's text.

Mutant 164

The mutation applied to mutant 164 caused the application to display a blank page when the user clicked on the "save" button without any changes being made to the form.

This case was not considered in the Capture/Replay script.

Regarding PBGT, and given that it generates a set of random paths of actions, the click-without-modifying test case was considered and so the failure was detected.

This is a failure that could possibly be detected by the Random testing approach; however, and given the specificity of the sequence of actions, it was not detected during the time we stipulated to perform the test.

Mutant 181

The mutant 181 disables the edit-user button presented in the table of users.

To access a user’s data edition page, the script generated from the PBGT’s model of the software goes first to the user’s profile page and only then clicks the edition button that is shown in there. Since it does not make use of the quick-access button existing in the table of users, it could not have detected the fault.

In the script of the Capture/Replay approach, the table button, which is located by an XPath expression, is clicked to proceed to edit a particular element. Given that the button was disabled, the error was detected.

3.8.2.4 Tudu Lists Particular Cases

Mutant 56

The mutant 56 always sets the date to the US date format, regardless of the selected date format, and even if no errors are displayed when selecting any of the other formats.

In the original non-mutated application, if the variable associated to this field was assigned with the value "null," or with a non-null value but different from the four available formats, the variable would be re-assigned with the value "US date format."

```
if (dateFormat == null || (!dateFormat.equals(Constants.DATEFORMAT_US)
                && !dateFormat.equals(Constants.DATEFORMAT_US_SHORT)
                && !dateFormat.equals(Constants.DATEFORMAT_FRENCH)
                && !dateFormat.equals(Constants.DATEFORMAT_FRENCH_SHORT)))
{dateFormat = Constants.DATEFORMAT_US;}
```

The introduced negation to the condition present in the if-statement makes the application to always assign the value "Constants.DATEFORMAT_US" to the variable "dateFormat" regardless of the option selected from the drop-down list.

The fault was not detected by the Capture/Replay testing approach. The test cases were designed to test the editing of user’s data, and after altering the fields (including changing the default date format), only an assertion to verify the presence of the message of success on editing after clicking on "submit" was inserted. Regarding other fields of the edit form, such as required fields, different test cases needed to be considered and a set of assertions was inserted; however, in the case of the data format, only the success message is considered since the alteration is done using a drop-down list, and no other information is provided in this page. Later, we could have checked in the application that the date of tasks really corresponded to this format. However, since in the recording session the selection of the "due date" was done using the calendars provided, which automatically inserted the date, this case passed unnoticed.

The PBGT testing approach detected the failure. The input text to define the “due date” is a string corresponding to the date in the format selected in the “information edit” form, which was defined as "valid." However, since the mutant was not able to change the date format, as required, the mutant interpreted that value as invalid and the test did not pass, being the fault detected.

3.8.3 Results Overview

It was found that the random testing approach was only able to find crashes – corroborating the assumptions – since that when a crash occurs no elements to interact with could be found, causing the testing tool to stop running. Such failures were detected between the time interval of 0 minutes, in cases where the application did not open or crashed after the first iteration, and 14 minutes for more specific situations that required a longer sequence of interactions.

However, and despite being the least capable of detecting different types of faults, it can be adopted if one only wants to find crashes; in fact, and excepting one specific mutant that requires a longer and more specific sequence of actions to crash, all the remaining crash failures were detected by this approach. Nevertheless, is must be said that the other approaches did also detect these, and given that they required less time to be "configured", it may not be the best choice.

In pair with PBGT, and not considering the time spent to learn the tool, the Capture/Replay approach presents a lower time regarding tests' implementation and execution, and was also able to find a considerable number of faults. However, it has its downsides. One of its biggest disadvantages is that it executes actions in a linear sequence, i.e., executes a certain set of defined actions always in the same sequence (corresponding to the sequence by which they were recorded during the script recording session). This is a disadvantage since there are errors that only occur when a certain action is performed before or after another one. Thus, this approach would not be able to detect these.

Also, by default a Capture/Replay tool identifies all the elements that do not have an associated text in the application (such as checkboxes) through their id. In the tested applications, there are elements added by the user while interacting with the application, for which id value is based on their position (index) in a table of the MySQL database. If the automatically generated script is not manually edited - keeping this locator - the database would have to be cleaned between each performed test.

Considering the characteristics of MySQL databases, this happens because even if an element is deleted from the application, the corresponding line on the database's table will be kept. Thus, when identifying elements this way there's a need to ensure that the initial state of the application – principally its database state – between each "script replay" is exactly the same. Since the Capture/Replay approach always inserts the elements in an established order, their position in the database will always be the same as well (considering that the software is functioning correctly).

However, this way of identifying elements could sometimes have the advantage of being capable to detect faults for which a test case was not specified. Let's consider, for instance, that a certain faulty application inserts multiple entries of a same element in the database. The index of the element to be nextly added by the test script would correspond to a higher value than the supposed one. Thus, and given that the script will select the element by its id, the application would return one of the repetitions of the first element and not the desired second one.

On the other hand, sometimes the id of an element is dynamic, which means that it would not be the same between tests and thus that it is not possible to locate the element by its id. In this situation, manual editing the script is absolutely required to insert a different locator.

Other particularity of this approach is that it is highly dependent on the specificity of the designed test cases and, if we want to cover a large amount of them, the task would become more time consuming and tedious, as described previously when justifying why this approach was not able to detect the malfunction of a specific contacts' filter. Moreover, it should be noted that posteriorly editing the script is not completely secure, since a specific sequence of actions was already defined; editing it may compromise the script's correct behavior.

PBGT was the approach that killed more mutants, detecting different faults; as a matter of fact, failures that could only be detected when performing actions in a specific sequence as well as failures related to invalid inputs were the ones that Capture/Replay failed the most to detect. However, it should be noted that none of the tested approaches has mechanisms to specifically verify the occurrence of repeated elements. In other words, there were cases where mutants showed various instances of a specific element in a drop-down list, for example, and these errors were not detected (not directly, at least).

The time taken to build the PARADIGM models is comparable to the one obtained in the capture/replay approach regarding building scripts; however, the PBGT approach takes longer to kill a mutant. Since we've chosen the "generate random paths" strategy, several test cases were generated and so, given all the possible combinations, it takes time to reach the one that leads the tool to detect a fault. Depending on the complexity of the model, the selection of the test strategy to generate the test script should be done carefully. A strategy that considers all the possible paths could lead to an explosion of the cases to be tested. All should be set depending on how important the test is as well as on the time available to perform it.

Chapter 4

Conclusions

This dissertation presented a comparative study of different GUI testing approaches. Having been established that they differ in many ways, we conducted a scientific experiment on those approaches in order to determine the most feasible or appropriate to test each software application.

To accomplish this, we proposed to compare them both in terms of effectiveness to detect failures and in terms of efforts required to test an application when adopting each of them. The metrics "mutation score" and time to build scripts/models and to kill a mutant were the ones used to evaluate results.

Then, we chose the applications to be tested and the tools by which the test would be performed. The former set was comprised of three web applications - iAddressBook, TaskFreak, and Tudu Lists; the latter included Sahi, PBGT, and three random testing programs developed by us since no GUI random testing tool was found to test web applications.

A detailed description of the approaches adopted to build the test scripts and to develop the random testing programs was provided. Then, and having the mutations been applied to our subjects (the applications to be tested), each mutant of each application was tested by each of the testing tools. The results were addressed in detail.

4.1 Goal Satisfaction

Our experiments allowed us to answer the proposed questions. It was found that the random testing approach was the least capable of detecting different types of faults and the one that took more time in activities prior to the testing per se. Only crashes were found when applying this approach, corroborating the assumptions; such crashes were detected in an acceptable interval of the time interval – between the 0 and 14 minutes (for situations that required a longer sequence of interactions).

The PBGT and the Capture/Replay approach showed similar results regarding the time to build the scripts/models. However, the former was able to find more faults, principally due to the facts that it generates random sequences of actions, contrary to the Capture/Replay approach, and that it features different mechanisms to verify results other than relying solely on assertions, like Capture/Replay does. Despite the good results, it should be mentioned that PBGT approach takes longer to kill a mutant, since among all the combinations of actions presented in the generated script, it could take time to reach the one that leads the tool to detect a fault.

Depending on the application, on the available time to build and execute the test scripts/programs, and on the type of faults a tester wants detect, the testing approach should be chosen carefully. With we hope this study can contribute on helping them making a decision when a GUI testing session is in perspective.

4.2 Future Work

The results of the experiment were clear regarding the characteristics of each approach in relation to time needed to test and failures detection, and the practical work exposed the benefits and drawbacks of each technique. However, and despite the results obtained, the experiment can still be improved.

The first limitation lies on the fact that only one person built the scripts and testing programs, which means that the results concerning the time taken to build them are solely based on the ones that person obtained. Since results could differ if the task was performed by others, as these are dependent on the tester's perception of the tested software as well as on his ability, the experiment should be performed by a variety of people so the results could be more accurate.

The second limitation relates to a possible lack of complexity regarding the selected subjects. It would be interesting to test more complex applications, that require bigger and more specific sequences of actions to show certain failures. We believe it could lead the results to be different regarding both time to build the script and the capacity to detect failures by the testing approaches; more specifically in what respects to the Capture/Replay, since the sequence of actions is defined by the tester.

References

[1] X. Yang, “Graphic user interface modelling and testing automation.” Victoria University, 2011.

[2] M. Andreessen, “Why Software Is Eating The World’,” *Wall Str. J.*, vol. 20, 2011.

[3] A. M. Memon, “A comprehensive framework for testing graphical user interfaces.” University of Pittsburgh, 2001.

[4] T. Hellmann, E. Moazzen, A. Sharma, M. Z. Akbar, J. Sillito, and F. Maurer, “An Exploratory Study of Automated GUI Testing: Goals, Issues, and Best Practices,” 2014.

[5] S. R. Ganov, C. Killmar, S. Khurshid, and D. E. Perry, “Test generation for graphical user interfaces based on symbolic execution,” in *Proceedings of the 3rd international workshop on Automation of software test*, 2008, pp. 33–40.

[6] B. A. Myers and M. B. Rosson, “Survey on User Interface Programming,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1992, pp. 195–202.

[7] B. N. Nguyen, B. Robbins, I. Banerjee, and A. Memon, “GUITAR: an innovative tool for automated testing of GUI-driven software,” *Autom. Softw. Eng.*, vol. 21, no. 1, pp. 65–105, 2013.

[8] P. Brooks, B. Robinson, and A. M. Memon, “An initial characterization of industrial graphical user interface systems,” in *Software Testing Verification and Validation, 2009. ICST’09. International Conference on*, 2009, pp. 11–20.

[9] S. Bauersfeld and T. E. J. Vos, “User interface level testing with TESTAR; what about more sophisticated action specification and selection?,” in *SATToSE*, 2014, pp. 60–78.

[10] “Automated GUI testing.” [Online]. Available: http://www.eecs.yorku.ca/course_archive/2006-07/W/4313/Slides/Module13-GUITesting.pdf.

[11] A. C. R. Paiva, “Automated Specification Based Testing of Graphical User Interfaces,” Faculty of Engineering, University of Porto, 2007.

[12] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro, “Automatic testing of GUI- based applications,” *Softw. Testing, Verif. Reliab.*, vol. 24, no. 5, pp. 341–366, 2014.

[13] V. Lelli, A. Blouin, and B. Baudry, “Classifying and qualifying GUI defects,” in

Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on, 2015, pp. 1–10.

[14] A. Memon, I. Banerjee, and A. Nagarajan, “What test oracle should I use for effective GUI testing?,” in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, 2003, pp. 164–173.

[15] “IBM Rational Help.” [Online]. Available: https://jazz.net/help-dev/clm/index.jsp?re=1&topic=/com.ibm.rational.test.qm.doc/topics/c_testcase_overview.html&scope=null. [Accessed: 12-Feb-2016].

[16] W. E. Lewis, *Software Testing and Continuous Quality Improvement, Third Edition*, 3rd ed. Boston, MA, USA: Auerbach Publications, 2008.

[17] W. E. Howden, “Reliability of the Path Analysis Testing Strategy,” *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 3, pp. 208–215, 1976.

[18] T. Y. Chen, T. H. Tse, and Z. Zhou, “Fault-based testing in the absence of an oracle,” in *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, 2001, pp. 172–178.

[19] E. Alégroth, “On the industrial applicability of visual gui testing,” *Dep. Comput. Sci. Eng. Softw. Eng. (Chalmers), Chalmers Univ. Technol. Goteborg, Tech. Rep*, 2013.

[20] X. Yuan, M. B. Cohen, and A. M. Memon, “GUI interaction testing: Incorporating event context,” *Softw. Eng. IEEE Trans.*, vol. 37, no. 4, pp. 559–574, 2011.

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

[22] S. A. Correia and A. R. Silva, “Técnicas para Construção de Testes Funcionais Automáticos.,” in *QUATIC*, 2004, pp. 111–117.

[23] K. Li and M. Wu, *Effective GUI testing automation: Developing an automated GUI testing tool*. John Wiley & Sons, 2006.

[24] A. M. Memon, M. E. Pollack, and M. Lou Soffa, “Using a goal-driven approach to generate test cases for GUIs,” in *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, 1999, pp. 257–266.

[25] S. Bauersfeld and T. E. J. Vos, “Advanced Monkey Testing for Real-World Applications.”

[26] W. Yang, Z. Chen, Z. Gao, Y. Zou, and X. Xu, “GUI testing assisted by human knowledge: Random vs. functional,” *J. Syst. Softw.*, vol. 89, pp. 76–86, 2014.

[27] B. Hofer, B. Peischl, and F. Wotawa, “Gui savvy end-to-end testing with smart monkeys,” in *Automation of Software Test, 2009. AST’09. ICSE Workshop on*, 2009, pp. 130–137.

[28] “Introduction to Test Monkey & CR tools.” [Online]. Available: http://www.cc.ntut.edu.tw/~wkchen/courses/wprog/wprog952/CRTools.pdf. [Accessed: 04-Feb-2016].

[29] A. Arcuri and L. Briand, “A practical guide for using statistical tests to assess randomized algorithms in software engineering,” in *Software Engineering (ICSE), 2011 33rd International Conference on*, 2011, pp. 1–10.

[30] V. Lelli, A. Blouin, B. Baudry, and F. Coulon, “On model-based testing advanced GUIs,” in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, 2015, pp. 1–10.

[31] R. M. L. M. Moreira and A. C. R. Paiva, “PBGT tool: an integrated modeling and testing

environment for pattern-based GUI testing," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 863–866.

[32] R. M. L. M. Moreira and A. C. R. Paiva, "A GUI modeling DSL for pattern-based GUI testing PARADIGM," in *Evaluation of Novel Approaches to Software Engineering (ENASE), 2014 International Conference on*, 2014, pp. 1–10.

[33] R. M. L. M. Moreira, A. C. R. Paiva, and A. Memon, "A pattern-based approach for gui modeling and testing," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, 2013, pp. 288–297.

[34] E. Alégroth, "Visual GUI Testing: Automating High-level Software Testing in Industrial Practice." Chalmers University of Technology, 2015.

[35] Z. Micskei, "The Gap Between Academic Research and Industrial Practice in Software Testing," 2014. [Online]. Available: http://mit.bme.hu/~micskeiz/papers/hustef-2014/micskei-hustef2014-material.pdf. [Accessed: 20-Jun-2002].

[36] S. Nedyalkova and J. Bernardino, "Comparative Study of Open Source Capture and Replay Tools," *Lat. Am. Trans. IEEE (Revista IEEE Am. Lat.*, vol. 12, no. 4, pp. 675–682, 2014.

[37] "HtmlUnit." [Online]. Available: http://htmlunit.sourceforge.net. [Accessed: 12-Feb-2016].

[38] E. Alégroth, "Random Visual GUI Testing: Proof of Concept.," in *SEKE*, 2013, pp. 178–183.

[39] "MonkeyRunner." [Online]. Available: http://developer.android.com/tools/help/monkeyrunner_concepts.html. [Accessed: 12-Feb-2016].

[40] J. Callahan, C. Roberts, M. J. Benson, N. Ye, A. C. Viars, and K. Kunderu, "Doit: Simple Web Application Testing." .

[41] F. A. Rosa, "WebTst." [Online]. Available: http://webtst.sourceforge.net/documentation.html. [Accessed: 12-Feb-2016].

[42] ThoughtWorks, "An introduction to Sahi," 2011. .

[43] S. Dutta, "Abbot-a friendly JUnit extension for GUI testing," *Java Dev. J.*, vol. 8, p. 12, 2003.

[44] T. Laurent, A. Ventresque, M. Papadakis, C. Henard, and Y. Le Traon, "Assessing and Improving the Mutation Testing Practice of {PIT}," *CoRR*, vol. abs/1601.0, 2016.

[45] R. Gopinath, C. Jensen, and A. Groce, "Mutations: How Close are they to Real Faults?," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 2014, pp. 189–200.

[46] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, 2011.

[47] S. Madiraju and S. R. A. J. Hurst, "Towards Automated Mutation Testing." 2004.

[48] K. Adamopoulos, M. Harman, and R. M. Hierons, "How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution," in *Genetic and Evolutionary Computation -- GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II*, K. Deb, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1338–1349.

[49] K. S. H. T. Wah, "An analysis of the coupling effect I: single test data," *Sci. Comput. Program.*, vol. 48, no. 2, pp. 119–161, 2003.

[50] S. B. Watson and L. James, "The scientific method: Is it still useful?," *Sci. Scope*, vol.

28, no. 3, pp. 37–39, 2004.

[51] D. M. Hillis, D. Sadava, H. C. Heller, and M. V. Price, *Principles of Life*. Sinauer Associates, 2010.

[52] R. V Blystone and K. Blodgett, “WWW: The Scientific Method,” *CBE— Life Sciences Education*, vol. 5, no. 1. pp. 7–11, 2006.

[53] M. Ryan and A. O’Callaghan, “The Scientific Method,” 2002.

[54] G. R. McPherson, “Teaching & Learning the Scientific Method,” *Am. Biol. Teach.*, vol. 63, no. 4, pp. 242–245, Apr. 2001.

[55] G. Dodig-Crnkovic, “Scientific Methods in Computer Science,” Mälardalen University.

[56] B. A. Schulte, “Scientific Writing & the Scientific Method: Parallel ‘Hourglass’ Structure in Form & Content,” *Am. Biol. Teach.*, vol. 65, no. 8, pp. 591–594, Oct. 2003.

[57] Y. SINGH and R. MALHOTRA, *OBJECT-ORIENTED SOFTWARE ENGINEERING*. PHI Learning, 2012.

[58] C. Wacha, “PHP iAddressBook,” 2015. [Online]. Available: http://iaddressbook.org/wiki/. [Accessed: 16-Apr-2016].

[59] S. Ozier, “TaskFreak - Original,” 2013. [Online]. Available: http://www.taskfreak.com/original. [Accessed: 16-Apr-2016].

[60] J. Dubois, “Tudu Lists,” 2016. [Online]. Available: http://www.julien-dubois.com/tudu-lists.html. [Accessed: 16-Apr-2016].

[61] Sahi, “Sahi - Assertions.” [Online]. Available: https://sahipro.com/docs/sahi-apis/assertions.html. [Accessed: 16-Apr-2016].

[62] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, “Is Lines of Code a Good Measure of Effort in Effort-aware Models?,” *Inf. Softw. Technol.*, vol. 55, no. 11, pp. 1981–1993, 2013.

[63] L. Prechelt, “An Empirical Comparison of Seven Programming Languages,” *Computer (Long. Beach. Calif).*, vol. 33, no. 10, pp. 23–29, 2000.

.

Apendix

A.1 Random Testing Programs

A.1.1 iAddressBook

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.UUID;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class IAddressBook {
	public static void main(String[] args) {

		// Create a new instance of the driver
		WebDriver driver = new FirefoxDriver();

		// Mut address
		driver.get("http://localhost:8888/iAddressBook");
		long t = System.currentTimeMillis() + 1800000;
		int i = 0;

		while (System.currentTimeMillis() < t) {
			// Find elements
			List<WebElement> allInputElements =
driver.findElements(By.xpath("//input[@type!='hidden' and @type!='file']"));
			List<WebElement> allLinkElements =
driver.findElements(By.xpath("//a[@href!='http://iaddressbook.org/' and
@href[not(contains(., 'export_vcard_cat')) and not(contains(.,
'export_csv_cat')) and not(contains(., 'export_ldif_cat')) and not(contains(.,
'import_folder'))]]"));
			List<WebElement> allTextareaElements =
driver.findElements(By.tagName("textarea"));
			List<WebElement> allSelectElements =
driver.findElements(By.tagName("select"));
			List<WebElement> allOptionElements =
driver.findElements(By.tagName("option"));

			List<WebElement> allElements = new
ArrayList<WebElement>();
			allElements.addAll(allInputElements);
			allElements.addAll(allLinkElements);
			allElements.addAll(allTextareaElements);
			allElements.addAll(allSelectElements);
			allElements.addAll(allOptionElements);

			Random randomGenerator = new Random();
```

```
                    int index = randomGenerator.nextInt(allElements.size());
                    WebElement element = allElements.get(index);
                    System.out.println(i + ". Selected element " +
element.getAttribute("outerHTML"));

                    if (element.isDisplayed()) {
                          switch (element.getTagName()) {
                          case "a":
                                if (element.getText().equals("delete
contact(s)")) {
                                      System.out.println("Clicked on element
" + element.getText());
                                      element.click();
                                      WebDriverWait wait = new
WebDriverWait(driver, 1);
                                      try {

      wait.until(ExpectedConditions.alertIsPresent());
                                            Alert alert =
driver.switchTo().alert();
                                            alert.accept();
                                      } catch (Exception e) {
                                      }
                                } else {
                                      System.out.println("Clicked on element
" + element.getText());
                                      element.click();
                                }
                                break;
                          case "input":
                                if
((element.getAttribute("type").equals("search")
                                            ||
element.getAttribute("type").equals("text"))) {
                                      if
(element.getAttribute("type").equals("search"))

      element.sendKeys(UUID.randomUUID().toString() + Keys.ENTER);
                                      else if
(element.getAttribute("name").contains("email"))

      element.sendKeys(UUID.randomUUID().toString() + "@email.com");
                                      else

      element.sendKeys(UUID.randomUUID().toString());
                                } else {
                                      if
(element.getAttribute("value").equals("Delete Contact")) {
                                            System.out.println("Clicked on
element <input>" + element.getText());
                                            element.click();
                                            WebDriverWait wait = new
WebDriverWait(driver, 1);
                                            try {

      wait.until(ExpectedConditions.alertIsPresent());
                                                  Alert alert =
driver.switchTo().alert();
                                                  alert.accept();
                                            } catch (Exception e) {
                                            }
                                      } else {
                                            System.out.println("Clicked on
element <input>" + element.getText());
                                            element.click();
```

```
                                                        }
                                                }
                                                break;
                                        case "textarea":

        element.sendKeys(UUID.randomUUID().toString());
                                                break;
                                        case "select":
                                                System.out.println("Clicked on element " +
element.getAttribute("outerHTML"));
                                                element.click();
                                                break;
                                        case "option":
                                                if (element.getText().equals("Custom...")) {
                                                        System.out.println("Clicked on element
" + element.getAttribute("outerHTML"));
                                                        element.click();
                                                        WebDriverWait wait = new
WebDriverWait(driver, 1);
                                                        try {

        wait.until(ExpectedConditions.alertIsPresent());
                                                                Alert alert =
driver.switchTo().alert();

        alert.sendKeys(UUID.randomUUID().toString());
                                                                alert.accept();
                                                        } catch (Exception e) {
                                                        }
                                                } else if
(element.getAttribute("value").contains("delcon") ||
element.getAttribute("value").contains("catdel")) {
                                                        System.out.println("Clicked on element
" + element.getAttribute("outerHTML"));
                                                        element.click();
                                                        WebDriverWait wait = new
WebDriverWait(driver, 1);
                                                        try {

        wait.until(ExpectedConditions.alertIsPresent());
                                                                Alert alert =
driver.switchTo().alert();
                                                                alert.accept();
                                                        } catch (Exception e) {
                                                        }
                                                } else {
                                                        System.out.println("Clicked on element
" + element.getAttribute("outerHTML"));
                                                        element.click();
                                                }
                                                break;
                                        }
                                }
                                i++;
                        }
                }
}
```

A.1.2 TaskFreak

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

Apendix

```
import java.util.UUID;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class TaskFreak {
	public static void main(String[] args) {

		// Create a new instance of the driver
		WebDriver driver = new FirefoxDriver();

		// Mut address
		driver.get("http://localhost:8888/TaskFreak/");
		long t = System.currentTimeMillis() + 1800000;
		int i = 0;

		while (System.currentTimeMillis() < t) {

			if (driver.getCurrentUrl().contains("login.php")) {
				WebElement elementt =
driver.findElement(By.xpath("//input[@name='username']"));
				elementt.sendKeys("admin");

				WebElement element2 =
driver.findElement(By.xpath("//input[@name='password']"));
				element2.sendKeys("");

				WebElement element3 =
driver.findElement(By.xpath("//input[@name='login']"));
				element3.click();
			} else {
				// Find elements
				List<WebElement> allInputElements =
driver.findElements(By.xpath("//input[@type!='hidden']"));
				List<WebElement> allLinkElements =
driver.findElements(By.xpath("//a[@href!='http://www.taskfreak.com']"));
				List<WebElement> allTextareaElements =
driver.findElements(By.tagName("textarea"));
				List<WebElement> allSelectElements =
driver.findElements(By.tagName("select"));
				List<WebElement> allOptionElements =
driver.findElements(By.tagName("option"));
				List<WebElement> allSortableElements =
driver.findElements(By.xpath("//th[@class='sortable']"));

				List<WebElement> allElements = new
ArrayList<WebElement>();
				allElements.addAll(allInputElements);
				allElements.addAll(allLinkElements);
				allElements.addAll(allTextareaElements);
				allElements.addAll(allSelectElements);
				allElements.addAll(allOptionElements);
				allElements.addAll(allSortableElements);

				Random randomGenerator = new Random();
				int index =
randomGenerator.nextInt(allElements.size());
				WebElement element = allElements.get(index);
				System.out.println(i+". Selected element " +
element.getAttribute("outerHTML"));

                           if (element.isDisplayed()) {
                                  switch (element.getTagName()) {
                                  case "a":
                                         if
(element.getAttribute("href").contains("freak_del") ||
element.getAttribute("href").contains("delete")) {
                                                System.out.println("Clicked on
element " + element.getAttribute("outerHTML"));
                                                element.click();

       driver.switchTo().alert().accept();
                                         } else {
                                                try {

       System.out.println("Clicked on element " +
element.getAttribute("outerHTML"));
                                                       element.click();
                                                } catch (Exception e) {

       System.out.println("Couldn't click on element: covered by another one");
                                                }
                                         }
                                         break;
                                  case "input":
                                         if
(element.getAttribute("type").equals("text")) {
                                                System.out.println("Sent keys to
element " + element.getAttribute("outerHTML"));

       element.sendKeys(UUID.randomUUID().toString());
                                         } else {
                                                if
(element.getAttribute("outerHTML").contains("return freak_sav()")) {

       System.out.println("Clicked on element " +
element.getAttribute("outerHTML"));
                                                       element.click();
                                                       WebDriverWait wait = new
WebDriverWait(driver, 1);
                                                       try{

wait.until(ExpectedConditions.alertIsPresent());
                                                           Alert alert =
driver.switchTo().alert();
                                                           alert.accept();
                                                       }
                                                       catch (Exception e){
                                                       }
                                                } else {

       System.out.println("Clicked on element " +
element.getAttribute("outerHTML"));
                                                       element.click();
                                                }
                                         }
                                         break;
                                  case "textarea":
                                         System.out.println("Sent keys to
element " + element.getAttribute("outerHTML"));

       element.sendKeys(UUID.randomUUID().toString());
                                         break;
                                  case "select":
                                  case "option":
                                  case "th":
                                         try {
                                                System.out.println("Clicked on
element " + element.getAttribute("outerHTML"));
                                                element.click();
                                         } catch (Exception e) {
                                                System.out.println("Couldn't
click on element: covered by another one");
                                         }
                                         break;
                                  }
                           } else {
                                  JavascriptExecutor executor =
(JavascriptExecutor) driver;
                                  System.out.println("Clicked on element " +
element.getAttribute("outerHTML"));

       executor.executeScript("arguments[0].click();", element);
                           }
                    }
                    i++;
             }
       }
}
```

A.1.3 Tudu Lists

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.UUID;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Tudu {
       public static void main(String[] args) {

              // Create a new instance of the driver
              WebDriver driver = new FirefoxDriver();

              // Mut address
              driver.get("http://localhost:8080/tudu-dwr/");
              long t = System.currentTimeMillis() + 1800000;
              int i = 0;

              while (System.currentTimeMillis() < t) {

                     if (driver.getCurrentUrl().contains("welcome.action")) {
                            WebElement elementt =
driver.findElement(By.xpath("//input[@name='j_username']"));
                            elementt.sendKeys("user");

                            WebElement element2 =
driver.findElement(By.xpath("//input[@name='j_password']"));
                            element2.sendKeys("test");
```

```
                           WebElement element3 = 
driver.findElement(By.xpath("//input[@value='Log In']"));
                           element3.click();
                    } else {
                           // Find elements
                           List<WebElement> allInputElements = 
driver.findElements(By.xpath("//input[@type!='hidden' and @type!='file']"));
                           List<WebElement> allLinkElements = 
driver.findElements(By.xpath("//a[@href!='http://tudu.sourceforge.net' and 
@href!='http://sourceforge.net/tracker/?group_id=131842' and 
@href[not(contains(., 'backupTodoList'))] and @href[not(contains(., 
'showRssFeed'))]]"));
                           List<WebElement> allTextareaElements = 
driver.findElements(By.tagName("textarea"));
                           List<WebElement> allSelectElements = 
driver.findElements(By.tagName("select"));
                           List<WebElement> allOptionElements = 
driver.findElements(By.tagName("option"));
                           List<WebElement> allSortableElements = 
driver.findElements(By.xpath("//th[@class[contains(., 'sort')]]"));
                           List<WebElement> allClickableTDElements = 
driver.findElements(By.xpath("//td[@onclick]"));

                           List<WebElement> allElements = new 
ArrayList<WebElement>();
                           allElements.addAll(allInputElements);
                           allElements.addAll(allLinkElements);
                           allElements.addAll(allTextareaElements);
                           allElements.addAll(allSelectElements);
                           allElements.addAll(allOptionElements);
                           allElements.addAll(allSortableElements);
                           allElements.addAll(allClickableTDElements);

                           Random randomGenerator = new Random();
                           int index = 
randomGenerator.nextInt(allElements.size());
                           WebElement element = allElements.get(index);
                           System.out.println(i + ". Selected element " + 
element.getAttribute("outerHTML"));

                           if (element.isDisplayed()) {
                                  switch (element.getTagName()) {
                                  case "a":
                                         if 
(element.getAttribute("href").contains("delete") 
                                                       || 
element.getAttribute("href").contains("addTodoListUser") 
                                                       || 
element.getText().equals("Submit")) {
                                                System.out.println("Clicked on 
element " + element.getAttribute("outerHTML"));
                                                try {
       
       System.out.println("Clicked on element " + 
element.getAttribute("outerHTML"));
                                                       element.click();
                                                } catch (Exception e) {
       
       System.out.println("Couldn't click on element: covered by another one");
                                                       break;
                                                }
                                                WebDriverWait wait = new 
WebDriverWait(driver, 1);
                                                try {
```

```
	wait.until(ExpectedConditions.alertIsPresent());
									Alert alert = 
driver.switchTo().alert();
									alert.accept();
								} catch (Exception e) {
								}
							} else {
								try {
									
	System.out.println("Clicked on element " + 
element.getAttribute("outerHTML"));
									element.click();
								} catch (Exception e) {
									
	System.out.println("Couldn't click on element: covered by another one");
								}
							}
							break;
						case "input":
							if 
(element.getAttribute("type").equals("text")
									|| 
element.getAttribute("type").equals("password")) {
								System.out.println("Sent keys to 
element " + element.getAttribute("outerHTML"));
								
	element.sendKeys(UUID.randomUUID().toString());
							} else {
								System.out.println("Clicked on 
element " + element.getAttribute("outerHTML"));
								element.click();
							}
							break;
						case "textarea":
							System.out.println("Sent keys to 
element " + element.getAttribute("outerHTML"));
							
	element.sendKeys(UUID.randomUUID().toString());
							break;
						case "select":
						case "option":
						case "th":
						case "td":
							try {
								System.out.println("Clicked on 
element " + element.getAttribute("outerHTML"));
								element.click();
							} catch (Exception e) {
								System.out.println("Couldn't 
click on element: covered by another one");
							}
							break;
						}
					} else
						System.out.println("Element " + 
element.getAttribute("outerHTML") + " is not visible");
				}
				i++;
			}
		}


}
```

A.2 Capture/Replay (Sahi) Scripts

A.2.1 iAddressBook

```
_click(_image("logo.png"));

//New contact 1
_click(_link("new contact"));
_setValue(_textbox("title"), "Mr.");
_setValue(_textbox("firstname"), "Jacob ");
_setValue(_textbox("lastname"), "Gyllenhaal");
_setValue(_textbox("nickname"), "Jake");
_setValue(_textbox("jobtitle"), "Actor");
_setValue(_textbox("department"), "Movies");
_setValue(_textbox("organization"), "Universal");
_setSelected(_select("phonelabel_1"), "work");
_setValue(_textbox("phone_1"), "911234567");
_setSelected(_select("emaillabel_1"), "work");
_setValue(_textbox("email_1"), "gyllen@email.com");
_setSelected(_select("urllabel_1"), "homepage");
_setValue(_textbox("url_1"), "http://www.website.com");
_setValue(_textbox("birthdate"), "1980-12-19");
_setSelected(_select("relatednamelabel_1"), "sister");
_setValue(_textbox("relatedname_1"), "Maggie Gyllenhaal");
_setSelected(_select("addresslabel_1"), "home");
_setValue(_textbox("street_1"), "Street Name");
_setValue(_textbox("city_1"), "Los Angeles");
_setValue(_textbox("state_1"), "California");
_setValue(_textbox("country_1"), "USA");
_setValue(_textarea("note"), "Notes Text");
_click(_submit("Save"));

//Verify entry
_click(_link("G"));
_assertExists(_link("Gyllenhaal, Jacob"));
_assert(_isVisible(_link("Gyllenhaal, Jacob")));
_assertEqual("Gyllenhaal, Jacob", _getText(_link("Gyllenhaal, Jacob")));
_assertContainsText("Gyllenhaal, Jacob", _link("Gyllenhaal, Jacob"));
```

```
//New contact 2
_click(_link("new contact"));
_setValue(_textbox("title"), "Mrs.");
_setValue(_textbox("firstname"), "Margalit");
_setValue(_textbox("firstname2"), "Ruth");
_setValue(_textbox("lastname"), "Gyllenhaal");
_setValue(_textbox("nickname"), "Maggie");
_setValue(_textbox("jobtitle"), "Actress");
_setValue(_textbox("department"), "Movies");
_setValue(_textbox("organization"), "Warner");
_setSelected(_select("phonelabel_1"), "work");
_setValue(_textbox("phone_1"), "911112233");
_setSelected(_select("emaillabel_1"), "work");
_setValue(_textbox("email_1"), "mgyllenhaal@email.com");
_setSelected(_select("urllabel_1"), "homepage");
_setValue(_textbox("url_1"), "http://www.website.com");
_setValue(_textbox("birthdate"), "1977-11-16");
_setSelected(_select("relatednamelabel_1"), "brother");
_setValue(_textbox("relatedname_1"), "Jake Gyllenhaal");
_setSelected(_select("addresslabel_1"), "home");
_setValue(_textbox("street_1"), "Street Name");
_setValue(_textbox("city_1"), "New York City");
_setValue(_textbox("state_1"), "New York");
_setValue(_textbox("country_1"), "USA");
_click(_submit("Save"));

//Verify entry
_click(_link("G"));
_assertExists(_link("Gyllenhaal, Margalit"));
_assert(_isVisible(_link("Gyllenhaal, Margalit")));
_assertEqual("Gyllenhaal, Margalit", _getText(_link("Gyllenhaal, Margalit")));
_assertContainsText("Gyllenhaal, Margalit", _link("Gyllenhaal, Margalit"));

//Edit contact 1
_click(_link("Gyllenhaal, Jacob"));
_assertExists(_div("Mr. Jacob Gyllenhaal"));
_assert(_isVisible(_div("Mr. Jacob Gyllenhaal")));
_assertEqual("Mr. Jacob Gyllenhaal", _getText(_div("Mr. Jacob Gyllenhaal")));
_assertContainsText("Mr. Jacob Gyllenhaal", _div("Mr. Jacob Gyllenhaal"));
```

```
_click(_submit("Edit"));
_setValue(_textbox("firstname2"), "Test");
_setValue(_textbox("lastname"), "Gyllen");
_click(_submit("Save"));

//Verify edition
_assertExists(_div("Mr. Jacob Test Gyllen"));
_assert(_isVisible(_div("Mr. Jacob Test Gyllen")));
_assertEqual("Mr. Jacob Test Gyllen", _getText(_div("Mr. Jacob Test Gyllen")));
_assertContainsText("Mr. Jacob Test Gyllen", _div("Mr. Jacob Test Gyllen"));
_click(_link("G"));
_assertExists(_link("Gyllen, Jacob"));
_assert(_isVisible(_link("Gyllen, Jacob")));
_assertEqual("Gyllen, Jacob", _getText(_link("Gyllen, Jacob")));
_assertContainsText("Gyllen, Jacob", _link("Gyllen, Jacob"));

//Test delete checkbox
_click(_link("new contact"));
_setValue(_textbox("firstname"), "Delete");
_setValue(_textbox("lastname"), "Contact");
_click(_submit("Save"));
_click(_link("C"));
_check(_checkbox(0, _leftOf(_link("Contact, Delete"))));
_expectConfirm("Do you want to delete the selected contacts?", true);
_click(_link("delete contact(s)"));
_assertEqual("Do you want to delete the selected contacts?", _lastConfirm());
_assertNotExists(_link("Contact, Delete"));
_assertNotVisible(_link("Contact, Delete"));

//Test delete button
_click(_link("new contact"));
_setValue(_textbox("firstname"), "Delete");
_setValue(_textbox("lastname"), "Contact 2");
_click(_submit("Save"));
_click(_link("Contact 2, Delete"));
_expectConfirm("Contact 2, Delete: Do you want to delete this contact?", true);
_click(_submit("Delete Contact"));
_assertEqual("Contact 2, Delete: Do you want to delete this contact?", _lastConfirm());
_click(_link("C"));
```

```
_assertNotExists(_link("Contact 2, Delete"));
_assertNotVisible(_link("Contact 2, Delete"));

_click(_image("logo.png"));
//New contact 1
_click(_link("new contact"));
_setValue(_textbox("title"), "Mr.");
_setValue(_textbox("firstname"), "Test");
_setValue(_textbox("lastname"), "Goodtest");
_setValue(_textbox("jobtitle"), "Tester");
_setValue(_textbox("department"), "Informatics");
_setValue(_textarea("category"), "Testing");
_click(_submit("Save"));

//Search for a non-existent contact
_setValue(_searchbox("q"), "non existent");
_typeKeyCodeNative(java.awt.event.KeyEvent.VK_ENTER);

//Verify that no contact is retrieved
_assertExists(_div("no contacts"));
_assert(_isVisible(_div("no contacts")));
_assertEqual("no contacts", _getText(_div("no contacts")));
_assertContainsText("no contacts", _div("no contacts"));



//Search and verify retrieval of contact's firstname
_setValue(_searchbox("q"), "test");
_typeKeyCodeNative(java.awt.event.KeyEvent.VK_ENTER);
_assertExists(_div("Mr. Test Goodtest"));
_assert(_isVisible(_div("Mr. Test Goodtest")));
_assertEqual("Mr. Test Goodtest", _getText(_div("Mr. Test Goodtest")));
_assertContainsText("Mr. Test Goodtest", _div("Mr. Test Goodtest"));

//Search for half of the contact's secondname
_setValue(_searchbox("q"), "good");
_typeKeyCodeNative(java.awt.event.KeyEvent.VK_ENTER);
_assertExists(_div("Mr. Test Goodtest"));
_assert(_isVisible(_div("Mr. Test Goodtest")));
_assertEqual("Mr. Test Goodtest", _getText(_div("Mr. Test Goodtest")));
```

```
_assertContainsText("Mr. Test Goodtest", _div("Mr. Test Goodtest"));

_click(_image("logo.png"));
//New contact 1 (with implicit creation of category)
_click(_link("new contact"));
_setValue(_textbox("firstname"), "Test");
_setValue(_textbox("lastname"), "One");
_setValue(_textarea("category"), "Test");
_click(_submit("Save"));

//New contact 2 (without category)
_click(_link("new contact"));
_setValue(_textbox("firstname"), "Test");
_setValue(_textbox("lastname"), "Two");
_click(_submit("Save"));

//New contact 3 (without category)
_click(_link("new contact"));
_setValue(_textbox("firstname"), "Test");
_setValue(_textbox("lastname"), "Three");
_click(_submit("Save"));

//Create Category 2
_setValue(_textbox("cat_name"), "Test2");
_click(_link("create category"));

//Add contact 3 to category 1 (implicitly created)
_check(_checkbox(0, _leftOf(_link("Three, Test"))));
_setSelected(_select("cat_menu"), "Test");

//Add contact 2 to category 2
_check(_checkbox(0, _leftOf(_link("Two, Test"))));
_setSelected(_select("cat_menu"), "Test2");

//Check category 1 contacts
_setSelected(_select("cat_id"), "Test");
_assertExists(_link("One, Test"));
_assertVisible(_link("One, Test"));
_assertEqual("One, Test", _getText(_link("One, Test")));
```

```
        _assertContainsText("One, Test", _link("One, Test"));
        _assertExists(_link("Three, Test"));
        _assertVisible(_link("Three, Test"));
        _assertEqual("Three, Test", _getText(_link("Three, Test")));
        _assertContainsText("Three, Test", _link("Three, Test"));

        //Delete contacts from category 1
        _check(_checkbox("selectall"));
        _expectConfirm("Do you want to delete the selected contacts?", true);
        _click(_link("delete contact(s)"));
        _assertEqual("Do you want to delete the selected contacts?", _lastConfirm());

        //Verify that category 1 is empty
        _setSelected(_select("cat_id"), "Test");
        _assertExists(_div("no contacts"));
        _assertVisible(_div("no contacts"));
        _assertEqual("no contacts", _getText(_div("no contacts")));
        _assertContainsText("no contacts", _div("no contacts"));

        //Check category 2 contacts
        _setSelected(_select("cat_id"), "Test2");
        _assertExists(_link("Two, Test"));
        _assertVisible(_link("Two, Test"));
        _assertEqual("Two, Test", _getText(_link("Two, Test")));
        _assertContainsText("Two, Test", _link("Two, Test"));

        //Remove contact from category (without deleting it)
        _setSelected(_select("cat_id"), "Test2");
        _check(_checkbox(0, _leftOf(_link("Two, Test"))));
        _expectConfirm("Do you want to remove the selected contacts from this category?", true);
        _setSelected(_select("cat_menu"), _byXPath("//option[contains(@value, 'delcon') and
text()='Test2']"));
        _assertEqual("Do you want to remove the selected contacts from this category?", _lastConfirm());

        //Verify that category 2 is empty and that contact 2 is still in contacts list
        _setSelected(_select("cat_id"), "Test2");
        _assertExists(_div("no contacts"));
        _assertVisible(_div("no contacts"));
        _assertEqual("no contacts", _getText(_div("no contacts")));
```

```
_assertContainsText("no contacts", _div("no contacts"));
_setSelected(_select("cat_id"), "All");
_assertExists(_link("Two, Test"));
_assertVisible(_link("Two, Test"));
_assertEqual("Two, Test", _getText(_link("Two, Test")));
_assertContainsText("Two, Test", _link("Two, Test"));

//Delete category 1
_setSelected(_select("cat_id"), "Test");
_expectConfirm("Do you want to delete this category?", true);
_setSelected(_select("cat_menu"), "delete category Test");
_assertNotExists(_option("Test", _in(_select("cat_id"))));
_assertNotVisible(_option("Test", _in(_select("cat_id"))));
```

A.2.2 TaskFreak

```
_click(_image("TaskFreak!"));

//Login - inexistent username
_setValue(_textbox("username"), "inexistentadmin");
_click(_submit("Login"));
_assertExists(_paragraph("box error"));
_assertVisible(_paragraph("box error"));
_assertEqual("Login Failed: username not found", _getText(_paragraph("box error")));
_assertContainsText("Login Failed: username not found", _paragraph("box error"));

//Login - wrong password
_setValue(_textbox("username"), "admin");
_setValue(_password("password"), "asdasd");
_click(_submit("Login"));
_assertExists(_paragraph("box error"));
_assertVisible(_paragraph("box error"));
_assertEqual("Login Failed: Wrong password", _getText(_paragraph("box error")));
_assertContainsText("Login Failed: Wrong password", _paragraph("box error"));

//Login and logout
_setValue(_textbox("username"), "admin");
_click(_submit("Login"));
_assertExists(_image("frk-logout"));
```

```
	_assertVisible(_image("frk-logout"));
	_click(_image("frk-logout"));
	_assertExists(_cell("You are now logged out. Goodbye."));
	_assertVisible(_cell("You are now logged out. Goodbye."));
	_assertEqual("You are now logged out. Goodbye.", _getText(_cell("You are now logged out.
Goodbye.")));
	_assertContainsText("You are now logged out. Goodbye.", _cell("You are now logged out.
Goodbye."));
	_click(_link("Click here to login"));
	_setValue(_textbox("username"), "admin");
	_click(_submit("Login"));

	//New user
	_click(_link("Users"));
	_click(_image("New"));
	_setValue(_textbox("firstName"), "Firstname");
	_setValue(_textbox("lastName"), "Lastname");
	_setValue(_textbox("username"), "User");
	_setValue(_password("password1"), "asdasd");
	_setValue(_password("password2"), "asdasd");
	_check(_checkbox("enabled"));
	_setSelected(_select("level"), "manager");
	_click(_submit("Create"));
	_assertExists(_cell("Firstname Lastname"));
	_assertVisible(_cell("Firstname Lastname"));
	_assertEqual("Firstname Lastname", _getText(_cell("Firstname Lastname")));
	_assertContainsText("Firstname Lastname", _cell("Firstname Lastname"));

	//Edit user - compulsory field "Firstname" empty
	_click(_link("Users"));
	_click(_image("b_edit.png[1]"));
	_setValue(_textbox("firstName"), "");
	_setValue(_password("password1"), "asdasd");
	_setValue(_password("password2"), "asdasd");
	_click(_submit("Save changes"));
	_assertExists(_span("This field is compulsory"));
	_assertVisible(_span("This field is compulsory"));
	_assertEqual("This field is compulsory", _getText(_span("This field is compulsory")));
	_assertContainsText("This field is compulsory", _span("This field is compulsory"));
```

Apendix

```
	//Edit user - Compulsory field "Lastname" empty
	_setValue(_textbox("firstName"), "Firstname");
	_setValue(_textbox("lastName"), "");
	_setValue(_password("password1"), "asdasd");
	_setValue(_password("password2"), "asdasd");
	_click(_submit("Save changes"));
	_assertExists(_span("This field is compulsory"));
	_assertVisible(_span("This field is compulsory"));
	_assertEqual("This field is compulsory", _getText(_span("This field is compulsory")));
	_assertContainsText("This field is compulsory", _span("This field is compulsory"));

	//Edit user - Compulsory field "Username" empty
	_setValue(_textbox("lastName"), "Lastname");
	_doubleClick(_textbox("username"));
	_setValue(_textbox("username"), "");
	_setValue(_password("password1"), "asdasd");
	_setValue(_password("password2"), "asdasd");
	_click(_submit("Save changes"));
	_assertExists(_span("username must have between 3 and 10 characters"));
	_assertVisible(_span("username must have between 3 and 10 characters"));
	_assertEqual("username must have between 3 and 10 characters", _getText(_span("username
must have between 3 and 10 characters")));
	_assertContainsText("username must have between 3 and 10 characters", _span("username must
have between 3 and 10 characters"));

	//Edit user
	_setValue(_textbox("title"), "Mr.");
	_setValue(_textbox("middleName"), "Middlename");
	_setValue(_textbox("email"), "email@email.com");
	_setValue(_textbox("city"), "P");
	_setSelected(_select("countryId"), "Portugal");
	_setValue(_textbox("username"), "User");
	_setValue(_password("password1"), "asdasd");
	_setValue(_password("password2"), "asdasd");
	_uncheck(_checkbox("enabled"));
	_click(_submit("Save changes"));
	_assertExists(_cell("Mr. Firstname M. Lastname"));
	_assertVisible(_cell("Mr. Firstname M. Lastname"));
```

```
_assertEqual("Mr. Firstname M. Lastname", _getText(_cell("Mr. Firstname M. Lastname")));
_assertContainsText("Mr. Firstname M. Lastname", _cell("Mr. Firstname M. Lastname"));
_assertExists(_span("Account is disabled!"));
_assertVisible(_span("Account is disabled!"));
_assertEqual("Account is disabled!", _getText(_span("Account is disabled!")));
_assertContainsText("Account is disabled!", _span("Account is disabled!"));
_click(_button("Back to list"));
_click(_image("New"));

//Create repeated user
_setValue(_textbox("firstName"), "Firstname2");
_setValue(_textbox("lastName"), "Lastname2");
_setValue(_textbox("username"), "user");
_setValue(_password("password1"), "asdasd");
_setValue(_password("password2"), "asdasd");
_click(_submit("Create"));
_assertExists(_span("username already exists"));
_assertVisible(_span("username already exists"));
_assertEqual("username already exists", _getText(_span("username already exists")));
_assertContainsText("username already exists", _span("username already exists"));
_click(_link("Users"));

//Delete user
_expectConfirm("really delete this user?", true);
_click(_image("b_dele.png[1]"));
_assertEqual("really delete this user?", _lastConfirm());
_assertNotExists(_link("Mr. Firstname M. Lastname"));
_assertNotVisible(_link("Mr. Firstname M. Lastname"));

_click(_link("Users"));
_click(_image("New"));
_setValue(_textbox("title"), "Mr.");
_setValue(_textbox("firstName"), "Tester");
_setValue(_textbox("firstName"), "Test");
_setValue(_textbox("lastName"), "Work");
_setValue(_textbox("username"), "tester");
_click(_submit("Create"));
_click(_button("Back to list"));
_expectConfirm("really enable this user?", true);
```

```
        _click(_image("b_disy.png"));
        _assertEqual("really enable this user?", _lastConfirm());

        //New project
        _click(_link("Projects"));
        _click(_image("New"));
        _setValue(_textbox("name"), "Project 1");
        _setValue(_textarea("description"), "New Project");
        _click(_submit("Create"));
        _assertExists(_link("Project 1[1]"));
        _assertVisible(_link("Project 1[1]"));
        _assertEqual("Project 1", _getText(_link("Project 1[1]")));
        _assertContainsText("Project 1", _link("Project 1[1]"));

        //Edit project - empty project name
        _click(_link("Projects"));
        _click(_image("b_edit.png"));
        _setValue(_textbox("name"), "");
        _setValue(_textarea("description"), "New Project");
        _click(_submit("Save"));
        _assertExists(_paragraph("box redish"));
        _assertVisible(_paragraph("box redish"));
        _assertEqual("There are some errors in the form - Information not saved!",
_getText(_paragraph("box redish")));
        _assertContainsText("There are some errors in the form - Information not saved!",
_paragraph("box redish"));

        //Edit project name, description and status
        _setValue(_textbox("name"), "Test Project 1");
        _setValue(_textarea("description"), "Proposal Project");
        _setSelected(_select("status"), "Proposal");
        _click(_submit("Save"));
        _assertExists(_link("Test Project 1[1]"));
        _assertVisible(_link("Test Project 1[1]"));
        _assertEqual("Test Project 1", _getText(_link("Test Project 1[1]")));
        _assertContainsText("Test Project 1", _link("Test Project 1[1]"));
        _click(_link("All Projects"));
        _assertNotExists(_cell("Test Project 1"));
        _assertNotVisible(_cell("Test Project 1"));
```

```
_click(_link("Projects"));
_click(_link("Test Project 1[1]"));
_assertExists(_textarea("description"));
_assertVisible(_textarea("description"));
_assertEqual("Proposal Project", _getValue(_textarea("description")));
_assertExists(_select("status"));
_assertVisible(_select("status"));
_assertEqual("Proposal", _getSelectedText(_select("status")));
_click(_paragraph("Add a user to this project"));
_click(_link("Add a user to this project"));
_setSelected(_select("nuser"), "Mr. Test Work");
_setSelected(_select("nposition"), "member");
_click(_submit("Add user to project"));
_click(_link("Test Project 1[1]"));
_assertExists(_link("Mr. Test Work"));
_assertVisible(_link("Mr. Test Work"));
_assertEqual("Mr. Test Work", _getText(_link("Mr. Test Work")));
_assertContainsText("Mr. Test Work", _link("Mr. Test Work"));
_click(_button("Back to list"));

//Delete project
_expectConfirm("Really delete this project and its tasks?", true);
_click(_image("Delete project"));
_assertEqual("Really delete this project and its tasks?", _lastConfirm());
_click(_link("Projects"));
_assertNotExists(_link("Test Project 1[1]"));
_assertNotVisible(_link("Test Project 1[1]"));

//Create task by clicking on image
_click(_image("TaskFreak!"));
_click(_image("New"));
_setSelected(_select("priority"), "Normal priority");
_setSelected(_select("context"), "Meeting");
_click(_image("Date selector"));
_click(_link("> new project?"));
_setValue(_textbox("project2"), "Thesis");
_setValue(_textbox("title"), "TaskFreak Tests");
_doubleClick(_textarea("description"));
_setValue(_textarea("description"), "Discuss advances");
```

```
_setSelected(_select("user"), "Mr. Test Work");
_setSelected(_select("status"), "20%");
_click(_submit("Save"));
_click(_link("All Users"));
_assertExists(_cell("TaskFreak Tests"));
_assertVisible(_cell("TaskFreak Tests"));
_assertEqual("TaskFreak Tests", _getText(_cell("TaskFreak Tests")));
_assertContainsText("TaskFreak Tests", _cell("TaskFreak Tests"));

//Create task by clicking on table link
_click(_image("TaskFreak!"));
_click(_link("New Todo"));
_setSelected(_select("priority"), "Medium priority");
_setSelected(_select("context"), "Other");
_click(_image("Date selector"));
_click(_link("> new project?"));
_setValue(_textbox("project2"), "CISTI");
_setValue(_textbox("title"), "Presentation");
_doubleClick(_textarea("description"));
_setValue(_textarea("description"), "Prepare presentation");
_setSelected(_select("user"), "Mr. Test Work");
_setSelected(_select("status"), "0%");
_check(_radio("showPrivate[2]"));
_click(_submit("Save"));
_click(_link("CISTI"));
_click(_link("Future Tasks[1]"));
_assertExists(_cell("Presentation"));
_assertVisible(_cell("Presentation"));
_assertEqual("Presentation", _getText(_cell("Presentation")));
_assertContainsText("Presentation", _cell("Presentation"));

//Edit by clicking on image
_click(_image("edit"));
_setValue(_textbox("title"), "Article Presentation");
_click(_submit("Save"));
_assertExists(_cell("Article Presentation"));
_assertVisible(_cell("Article Presentation"));
_assertEqual("Article Presentation", _getText(_cell("Article Presentation")));
_assertContainsText("Article Presentation", _cell("Article Presentation"));
```

```
//Edit by clicking on table image
_click(_link("All Projects"));
_click(_cell("Thesis"));
_click(_image("edit"));
_doubleClick(_textbox("title"));
_setValue(_textbox("title"), "Applications Tests");
_setValue(_textarea("description"), "Show and discuss advances");
_click(_submit("Save"));
_click(_link("All Users"));
_assertExists(_cell("Applications Tests"));
_assertVisible(_cell("Applications Tests"));
_assertEqual("Applications Tests", _getText(_cell("Applications Tests")));
_assertContainsText("Applications Tests", _cell("Applications Tests"));

//Create one more task for the same project
_click(_image("TaskFreak!"));
_click(_image("New"));
_setSelected(_select("priority"), "Medium priority");
_setSelected(_select("context"), "Document");
_setSelected(_select("project"), "Thesis");
_setValue(_textbox("title"), "Describe differences");
_doubleClick(_textarea("description"));
_setValue(_textarea("description"), "Describe differences between CR and PBGT results");
_click(_submit("Save"));
_click(_link("All Tasks[2]"));
_assertExists(_cell("Describe differences"));
_assertVisible(_cell("Describe differences"));
_assertEqual("Describe differences", _getText(_cell("Describe differences")));
_assertContainsText("Describe differences", _cell("Describe differences"));
_assertExists(_cell("Applications Tests"));
_assertVisible(_cell("Applications Tests"));
_assertEqual("Applications Tests", _getText(_cell("Applications Tests")));
_assertContainsText("Applications Tests", _cell("Applications Tests"));

//Delete task
_expectConfirm("Really delete this task?", true);
_click(_image("del[1]"));
_assertEqual("Really delete this task?", _lastConfirm());
```

```
//Change task completion status
_click(_image("edit"));
_setSelected(_select("status"), "20%");
_click(_submit("Save"));
_click(_cell("Thesis"));
_assertExists(_div("20%"));
_assertVisible(_div("20%"));
_assertEqual("20%", _getText(_div("20%")));
_assertContainsText("20%", _div("20%"));
_click(_image("close"));
_click(_cell("est36"));
_click(_cell("Thesis"));
_assertExists(_div("60%"));
_assertVisible(_div("60%"));
_assertEqual("60%", _getText(_div("60%")));
_assertContainsText("60%", _div("60%"));
_click(_image("close"));

//Comments management
_click(_link("All Projects"));
_click(_image("commentaires[1]"));
_click(_link("post first comment"));
_setValue(_textarea("veditbody"), "New comment");
_click(_submit("Save"));
_click(_div("vmore"));
_click(_image("close"));
_click(_cell("CISTI"));
_click(_link("comments"));
_assertExists(_div("New comment"));
_assertVisible(_div("New comment"));
_assertEqual("New comment", _getText(_div("New comment")));
_assertContainsText("New comment", _div("New comment"));
_click(_link("edit[1]"));
_setValue(_textarea("veditbody"), "New comment edited");
_click(_submit("Save"));
_click(_image("close"));
_click(_cell("CISTI"));
_click(_link("comments"));
```

```
        _assertExists(_div("New comment edited"));
        _assertVisible(_div("New comment edited"));
        _assertEqual("New comment edited", _getText(_div("New comment edited")));
        _assertContainsText("New comment edited", _div("New comment edited"));
        _expectConfirm("really delete comment?", true);
        _click(_link("delete[1]"));
        _assertEqual("really delete comment?", _lastConfirm());
        _click(_image("close"));
        _click(_image("commentaires[1]"));
        _assertExists(_div("-no comment left yet-post first comment"));
        _assertVisible(_div("-no comment left yet-post first comment"));
        _assertEqual("-no comment left yet-post first comment", _getText(_div("-no comment left yet-
post first comment")));
        _assertContainsText("-no comment left yet-post first comment", _div("-no comment left yet-post
first comment"));
        _click(_image("close"));

        //Filter by user and context
        _click(_link("Projects"));
        _click(_image("b_edit.png"));
        _click(_link("Add a user to this project"));
        _setSelected(_select("nposition"), "member");
        _click(_submit("Add user to project"));
        _click(_image("TaskFreak!"));
        _click(_image("edit[1]"));
        _setSelected(_select("user"), "Mr. Test Work");
        _click(_submit("Save"));
        _setSelected(_select("sUser"), "All Users");
        _assertExists(_cell("CISTI"));
        _assertVisible(_cell("CISTI"));
        _assertEqual("CISTI", _getText(_cell("CISTI")));
        _assertContainsText("CISTI", _cell("CISTI"));
        _assertExists(_cell("Thesis"));
        _assertVisible(_cell("Thesis"));
        _assertEqual("Thesis", _getText(_cell("Thesis")));
        _assertContainsText("Thesis", _cell("Thesis"));
        _setSelected(_select("sUser"), "Admin");
        _assertExists(_cell("Thesis"));
        _assertVisible(_cell("Thesis"));
        _assertEqual("Thesis", _getText(_cell("Thesis")));
        _assertContainsText("Thesis", _cell("Thesis"));
        _assertNotExists(_cell("CISTI"));
        _assertNotVisible(_cell("CISTI"));
        _setSelected(_select("sUser"), "Test");
        _assertExists(_cell("CISTI"));
        _assertVisible(_cell("CISTI"));
        _assertEqual("CISTI", _getText(_cell("CISTI")));
        _assertContainsText("CISTI", _cell("CISTI"));
        _assertNotExists(_cell("Thesis"));
        _assertNotVisible(_cell("Thesis"));
        _setSelected(_select("sUser"), "All Users");
        _setSelected(_select("sContext"), "Meeting");
        _assertExists(_paragraph("- no task match your criterions -"));
        _assertVisible(_paragraph("- no task match your criterions -"));
        _assertEqual("- no task match your criterions -", _getText(_paragraph("- no task match your
criterions -")));
        _assertContainsText("- no task match your criterions -", _paragraph("- no task match your
criterions -"));
        _setSelected(_select("sContext"), "Document");
        _assertExists(_cell("Thesis"));
        _assertVisible(_cell("Thesis"));
        _assertEqual("Thesis", _getText(_cell("Thesis")));
        _assertContainsText("Thesis", _cell("Thesis"));
        _assertNotExists(_cell("CISTI"));
        _assertNotVisible(_cell("CISTI"));
        _setSelected(_select("sContext"), "Other");
        _assertExists(_cell("CISTI"));
        _assertVisible(_cell("CISTI"));
        _assertEqual("CISTI", _getText(_cell("CISTI")));
        _assertContainsText("CISTI", _cell("CISTI"));
        _assertNotExists(_cell("Thesis"));
        _assertNotVisible(_cell("Thesis"));
        _setSelected(_select("sContext"), "All Contexts");
        _click(_link("All Projects"));

        //Sort by task name
        _click(_tableHeader("Project"));
        _assertEqual("CISTI", _getText(_cell(_table("taskSheet"), 1, 2)))
```

```
	_click(_tableHeader("Project"));
	_assertEqual("CISTI", _getText(_cell(_table("taskSheet"), 5, 2)))

	//Sort by user name
	_click(_tableHeader("User"));
	_assertEqual("Test", _getText(_cell(_table("taskSheet"), 5, 4)))
	_click(_tableHeader("Project"));
	_assertEqual("Test", _getText(_cell(_table("taskSheet"), 1, 4)))
```

A.2.3 Tudu Lists

```
	_click(_link("Welcome"));

	//Nonexistent user
	_setValue(_textbox("j_username"), "nonexistent");
	_setValue(_password("j_password"), "nonexistentp");
	_click(_submit("Log In"));
	_assertExists(_div("Your login attempt was not successful, please try again."));
	_assertVisible(_div("Your login attempt was not successful, please try again."));
	_assertEqual("Your login attempt was not successful, please try again.", _getText(_div("Your login
attempt was not successful, please try again.")));
	_assertContainsText("Your login attempt was not successful, please try again.", _div("Your login
attempt was not successful, please try again."));

	//Only user
	_setValue(_textbox("j_username"), "user");
	_setValue(_password("j_password"), "");
	_click(_submit("Log In"));
	_assertExists(_div("Your login attempt was not successful, please try again."));
	_assertVisible(_div("Your login attempt was not successful, please try again."));
	_assertEqual("Your login attempt was not successful, please try again.", _getText(_div("Your login
attempt was not successful, please try again.")));
	_assertContainsText("Your login attempt was not successful, please try again.", _div("Your login
attempt was not successful, please try again."));

	//Only pass
	_setValue(_textbox("j_username"), "");
	_setValue(_password("j_password"), "test");
	_click(_submit("Log In"));
	_assertExists(_div("Your login attempt was not successful, please try again."));
	_assertVisible(_div("Your login attempt was not successful, please try again."));
	_assertEqual("Your login attempt was not successful, please try again.", _getText(_div("Your login attempt was not successful, please try again.")));
	_assertContainsText("Your login attempt was not successful, please try again.", _div("Your login attempt was not successful, please try again."));

	//Login
	_setValue(_textbox("j_username"), "user");
	_setValue(_password("j_password"), "test");
	_click(_submit("Log In"));
	_assertExists(_link("My Todos"));
	_assertVisible(_link("My Todos"));
	_assertEqual("My Todos", _getText(_link("My Todos")));
	_assertContainsText("My Todos", _link("My Todos"));

	//Logout
	_click(_link("Log out"));
	_assertExists(_div("You have left Tudu Lists. Click here if you want to reconnect"));
	_assertVisible(_div("You have left Tudu Lists. Click here if you want to reconnect"));
	_assertEqual("You have left Tudu Lists. Click here if you want to reconnect", _getText(_div("You have left Tudu Lists. Click here if you want to reconnect")));
	_assertContainsText("You have left Tudu Lists. Click here if you want to reconnect", _div("You have left Tudu Lists. Click here if you want to reconnect"));

	//Edit profile
	_click(_link("Click here if you want to reconnect"));
	_setValue(_textbox("j_username"), "user");
	_setValue(_password("j_password"), "test");
	_click(_submit("Log In"));
	_click(_link("My info"));
	_assertExists(_heading3("Manage user information"));
	_assertVisible(_heading3("Manage user information"));
	_assertEqual("Manage user information", _getText(_heading3("Manage user information")));
	_assertContainsText("Manage user information", _heading3("Manage user information"));

	//Empty fields
	_setValue(_textbox("firstName"), "");
```

```
_click(_submit("Submit"));
_assertExists(_div("First name is required."));
_assertVisible(_div("First name is required."));
_assertEqual("First name is required.", _getText(_div("First name is required.")));
_assertContainsText("First name is required.", _div("First name is required."));

_setValue(_textbox("firstName"), "User");
_setValue(_textbox("lastName"), "");
_click(_submit("Submit"));
_assertExists(_div("Last name is required."));
_assertVisible(_div("Last name is required."));
_assertEqual("Last name is required.", _getText(_div("Last name is required.")));
_assertContainsText("Last name is required.", _div("Last name is required."));

//Edit
_setValue(_textbox("firstName"), "Edited User");
_setValue(_textbox("lastName"), "To Test");
_setSelected(_select("dateFormat"), "dd/mm/yyyy");
_setValue(_password("password"), "newtest");
_setValue(_password("verifyPassword"), "newtest");
_click(_submit("Submit"));
_assertExists(_span("Information saved."));
_assertVisible(_span("Information saved."));
_assertEqual("Information saved.", _getText(_span("Information saved.")));
_assertContainsText("Information saved.", _span("Information saved."));

_click(_link("My Todos"));
//Add lists
_click(_link("Add a new list"));
_setValue(_textbox("name"), "New List");
_click(_link("Submit[2]"));
_assertExists(_link("New List (0/0)"));
_assertVisible(_link("New List (0/0)"));
_assertEqual("New List (0/0)", _getText(_link("New List (0/0)")));
_assertContainsText("New List (0/0)", _link("New List (0/0)"));
_click(_link("Add a new list"));
_setValue(_textbox("name"), "New List 2");
_check(_checkbox("rssAllowed"));
_click(_link("Submit[2]"));
```

```
_assertExists(_link("New List 2 (0/0)"));
_assertVisible(_link("New List 2 (0/0)"));
_assertEqual("New List 2 (0/0)", _getText(_link("New List 2 (0/0)")));
_assertContainsText("New List 2 (0/0)", _link("New List 2 (0/0)"));

//Edit list
_click(_link("New List 2 (0/0)"));
_assertExists(_div("New List 2"));
_assertVisible(_div("New List 2"));
_assertEqual("New List 2", _getText(_div("New List 2")));
_assertContainsText("New List 2", _div("New List 2"));
_click(_link("Edit current list"));
_setValue(_textbox("name[1]"), "New List 2 Edited");
_uncheck(_checkbox("rssAllowed[1]"));
_click(_link("Submit[3]"));
_assertExists(_link("New List 2 Edited (0/0)"));
_assertVisible(_link("New List 2 Edited (0/0)"));
_assertEqual("New List 2 Edited (0/0)", _getText(_link("New List 2 Edited (0/0)")));
_assertContainsText("New List 2 Edited (0/0)", _link("New List 2 Edited (0/0)"));

//Delete list
_click(_link("New List (0/0)"));
_expectConfirm("Are you sure you want to delete this Todo List?", true);
_click(_link("Delete current list"));
_assertEqual("Are you sure you want to delete this Todo List?", _lastConfirm());
_assertExists(_div("Todo List successfully deleted."));
_assertVisible(_div("Todo List successfully deleted."));
_assertEqual("Todo List successfully deleted.", _getText(_div("Todo List successfully deleted.")));
_assertContainsText("Todo List successfully deleted.", _div("Todo List successfully deleted."));
_assertNotExists(_link("New List (0/0)"));
_assertNotVisible(_link("New List (0/0)"));

//Quick add
_click(_link("New List 2 Edited (0/0)"));
_assertExists(_cell("(100%)"));
_assertVisible(_cell("(100%)"));
_assertEqual("(100%)", _getText(_cell("(100%)")));
_assertContainsText("(100%)", _cell("(100%)"));
_setValue(_textbox("description[2]"), "Write random tests description");
```

```
	_click(_link("Quick Add"));
	_assertExists(_div("Write random tests description"));
	_assertVisible(_div("Write random tests description"));
	_assertEqual("Write random tests description", _getText(_div("Write random tests description")));
	_assertContainsText("Write random tests description", _div("Write random tests description"));
	_assertExists(_cell("(0%)"));
	_assertVisible(_cell("(0%)"));
	_assertEqual("(0%)", _getText(_cell("(0%)")));
	_assertContainsText("(0%)", _cell("(0%)"));

	//Advanced add
	_setValue(_textbox("description[2]"), "Todo 2");
	_click(_link("Advanced Add"));
	_setValue(_textbox("description"), "Thesis advance");
	_setValue(_textbox("priority"), "N");
	_click(_image("Calendar"));
	_setSelected(_select("assignedUser"), "user");
	_setValue(_textarea("notes"), "Tests results");
	_click(_link("Submit"));
	_assertEqual("Validation error : the priority is not a number.", _lastAlert());
	_setValue(_textbox("priority"), "1");
	_click(_link("Submit"));
	_assertExists(_div("Thesis advance"));
	_assertVisible(_div("Thesis advance"));
	_assertEqual("Thesis advance", _getText(_div("Thesis advance")));
	_assertContainsText("Thesis advance", _div("Thesis advance"));

	//Edit
	_click(_link("My Todos"));
	_click(_div("Thesis advance"));
	_setValue(_textbox("/edit-in-/"), "Thesis advances (description edited)");
	_click(_div("todosTable"));
	_assertExists(_div("Thesis advances (description edited)"));
	_assertVisible(_div("Thesis advances (description edited)"));
	_assertEqual("Thesis advances (description edited)", _getText(_div("Thesis advances (description
edited)")));
	_assertContainsText("Thesis advances (description edited)", _div("Thesis advances (description
edited)"));
```

```
_click(_image("pencil.png[2]"));
_setValue(_textbox("description[1]"), "Write tests");
_setValue(_textarea("notes[1]"), "Notes edited");
_setValue(_textbox("priority[1]"), "4");
_setSelected(_select("assignedUser[1]"), "-- Not assigned --");
_click(_link("Submit[1]"));
_assertExists(_div("Write tests"));
_assertVisible(_div("Write tests"));
_assertEqual("Write tests", _getText(_div("Write tests")));
_assertContainsText("Write tests", _div("Write tests"));
_assertExists(_cell("4"));
_assertVisible(_cell("4"));
_assertEqual("4", _getText(_cell("4")));
_assertContainsText("4", _cell("4"));
_click(_link("Assigned to me"));
_assertNotExists(_row("New List 2 Edited Write tests 4"));
_assertNotVisible(_row("New List 2 Edited Write tests 4"));
_click(_link("New List 2 Edited (0/2)"));

_setValue(_textbox("description[2]"), "Todo to delete");
_click(_link("Quick Add"));
_setValue(_textbox("description[2]"), "Todo to test filter");
_click(_link("Quick Add"));
_setValue(_textbox("description[2]"), "Todo to test filter 2");
_click(_link("Quick Add"));
_click(_image("pencil.png[5]"));
_setValue(_textbox("priority[1]"), "2");
_setValue(_textbox("dueDate[1]"), "14/05/2015");
_setValue(_textarea("notes[1]"), "Old todo");
_click(_link("Submit[1]"));

//Delete
_click(_image("pencil.png[5]"));
_setValue(_textbox("priority[1]"), "4");
_setValue(_textbox("dueDate[1]"), "18/05/2016");
_click(_link("Submit[1]"));
_expectConfirm("Are you sure you want to delete this Todo?", true);
_click(_image("bin_closed.png[5]"));
_assertEqual("Are you sure you want to delete this Todo?", _lastConfirm());
```

```
_assertNotExists(_div("Todo to delete"));
_assertNotVisible(_div("Todo to delete"));

//Complete
_click(_image("pencil.png[2]"));
_setValue(_textbox("dueDate[1]"), "04/06/2016");
_click(_link("Submit[1]"));
_check(_checkbox("on[3]"));
_assertExists(_link("New List 2 Edited (1/4)"));
_assertVisible(_link("New List 2 Edited (1/4)"));
_assertEqual("New List 2 Edited (1/4)", _getText(_link("New List 2 Edited (1/4)")));
_assertContainsText("New List 2 Edited (1/4)", _link("New List 2 Edited (1/4)"));
_assertExists(_cell("(25%)"));
_assertVisible(_cell("(25%)"));
_assertEqual("(25%)", _getText(_cell("(25%)")));
_assertContainsText("(25%)", _cell("(25%)"));

//Show todos
_assertExists(_cell("0 hidden Todo(s)."));
_assertVisible(_cell("0 hidden Todo(s)."));
_assertEqual("0 hidden Todo(s).", _getText(_cell("0 hidden Todo(s).")));
_assertContainsText("0 hidden Todo(s).", _cell("0 hidden Todo(s)."));
_click(_link("Show older Todos"));
_assertExists(_link("Hide older Todos"));
_assertVisible(_link("Hide older Todos"));
_assertEqual("Hide older Todos", _getText(_link("Hide older Todos")));
_assertContainsText("Hide older Todos", _link("Hide older Todos"));
_assertNotExists(_cell("0 hidden Todo(s)."));
_assertNotVisible(_cell("0 hidden Todo(s)."));
_click(_link("Hide older Todos"));

//Delete completed
_expectConfirm("Are you sure you want to delete all the completed Todos?", true);
_click(_link("Delete completed Todos"));
_assertEqual("Are you sure you want to delete all the completed Todos?", _lastConfirm());
_assertNotExists(_div("Write tests"));
_assertNotVisible(_div("Write tests"));
_assertExists(_link("New List 2 Edited (0/3)"));
_assertVisible(_link("New List 2 Edited (0/3)"));
```

```
        _assertEqual("New List 2 Edited (0/3)", _getText(_link("New List 2 Edited (0/3)")));
        _assertContainsText("New List 2 Edited (0/3)", _link("New List 2 Edited (0/3)"));

        //Filter
        _assertExists(_div("Todo to test filter"));
        _assertVisible(_div("Todo to test filter"));
        _assertEqual("Todo to test filter", _getText(_div("Todo to test filter")));
        _assertContainsText("Todo to test filter", _div("Todo to test filter"));
        _assertExists(_div("Todo to test filter 2"));
        _assertVisible(_div("Todo to test filter 2"));
        _assertEqual("Todo to test filter 2", _getText(_div("Todo to test filter 2")));
        _assertContainsText("Todo to test filter 2", _div("Todo to test filter 2"));
        _assertExists(_div("Thesis advances (description edited)"));
        _assertVisible(_div("Thesis advances (description edited)"));
        _assertEqual("Thesis advances (description edited)", _getText(_div("Thesis advances (description
edited)")));
        _assertContainsText("Thesis advances (description edited)", _div("Thesis advances (description
edited)"));
        _check(_checkbox("on[3]"));

        _click(_link("Next 4 days"));
        _assertExists(_div("Todos for the next 4 days"));
        _assertVisible(_div("Todos for the next 4 days"));
        _assertEqual("Todos for the next 4 days", _getText(_div("Todos for the next 4 days")));
        _assertContainsText("Todos for the next 4 days", _div("Todos for the next 4 days"));
        _assertExists(_cell("Todo to test filter 2"));
        _assertVisible(_cell("Todo to test filter 2"));
        _assertEqual("Todo to test filter 2", _getText(_cell("Todo to test filter 2")));
        _assertContainsText("Todo to test filter 2", _cell("Todo to test filter 2"));
        _assertNotExists(_cell("Todo to test filter"));
        _assertNotVisible(_cell("Todo to test filter"));
        _assertNotExists(_div("Thesis advances (description edited)"));
        _assertNotVisible(_div("Thesis advances (description edited)"));

        _click(_link("Assigned to me"));
        _assertExists(_div("Todos assigned to me"));
        _assertVisible(_div("Todos assigned to me"));
        _assertEqual("Todos assigned to me", _getText(_div("Todos assigned to me")));
        _assertContainsText("Todos assigned to me", _div("Todos assigned to me"));
        _assertExists(_cell("Todo to test filter 2"));
        _assertVisible(_cell("Todo to test filter 2"));
        _assertEqual("Todo to test filter 2", _getText(_cell("Todo to test filter 2")));
        _assertContainsText("Todo to test filter 2", _cell("Todo to test filter 2"));
        _assertExists(_cell("Thesis advances (description edited)"));
        _assertVisible(_cell("Thesis advances (description edited)"));
        _assertEqual("Thesis advances (description edited)", _getText(_cell("Thesis advances (description
edited)")));
        _assertContainsText("Thesis advances (description edited)", _cell("Thesis advances (description
edited)"));
```

A.3 Tables of Tests Resuts

A.3.1 iAddressBook

Mutant no.	Mutated File	Line no.	Original line	Modified line	Random	Capture/Replay	PBGT
11	actions.php	233	if($contact == false)	if($contact != false)	Killed	Killed	Killed
28		431	===	!==	Alive	Killed	Killed
29		448	!TRUE	TRUE	Alive	Killed	Killed
30		458	TRUE	!TRUE	Alive	Killed	Killed
31		460	==	!=	Alive	Killed	Killed
32		463	TRUE	!TRUE	Alive	Alive	Killed
33		467	==	!=	Alive	Killed	Killed
34		470	TRUE	!TRUE	Alive	Alive	Killed
35		477	!=	==	Alive	Killed	Killed
36		497	TRUE	!TRUE	Alive	Killed	Killed
37		508	TRUE	!TRUE	Alive	Killed	Killed
38		510	===	!==	Alive	Killed	Killed
41		551	===	!==	Alive	Killed	Alive
43		580	TRUE	!TRUE	Killed	Killed	Killed
44	addressbook.php	25	TRUE	!TRUE	Alive	Killed	Killed
45		27	TRUE	!TRUE	Alive	Killed	Killed
46		31	!TRUE	TRUE	Alive	Killed	Killed
47		74	!TRUE	TRUE	Alive	Killed	Killed
48		86	==	!=	Alive	Killed	Killed
49		146	TRUE	!TRUE	Alive	Killed	Killed
50		147	TRUE	!TRUE	Killed	Killed	Killed
61		262	==	!=	Alive	Killed	Killed
62		309	!TRUE	TRUE	Alive	Killed	Killed
66		334	!TRUE	TRUE	Alive	Killed	Killed
67		343	!TRUE	TRUE	Alive	Alive	Killed
71		432	TRUE	!TRUE	Alive	Killed	Killed
72	category.php	9	!TRUE	TRUE	Alive	Killed	Killed
73		54	TRUE	!TRUE	Alive	Killed	Killed

74		56	TRUE	!TRUE	Alive	Killed	Killed
75		60	!TRUE	TRUE	Alive	Killed	Killed
76		70	==	!=	Alive	Killed	Killed
77		71	==	!=	Alive	Killed	Killed
78		72	==	!=	Alive	Killed	Killed
82		105	!TRUE	TRUE	Alive	Killed	Killed
83		107	TRUE	!TRUE	Alive	Killed	Killed
84		115	==	!=	Alive	Killed	Killed
85		134	!TRUE	TRUE	Alive	Killed	Killed
87		149	!TRUE	TRUE	Alive	Killed	Killed
88		156	!TRUE	TRUE	Alive	Alive	Killed
89		163	!TRUE	TRUE	Alive	Alive	Killed
90		175	!TRUE	TRUE	Alive	Alive	Killed
91		191	TRUE	!TRUE	Alive	Alive	Killed
92		192	TRUE	!TRUE	Killed	Killed	Alive
93		207	!TRUE	TRUE	Alive	Killed	Killed
94		215	TRUE	!TRUE	Alive	Killed	Killed
95		216	TRUE	!TRUE	Alive	Killed	Killed
96		236	!TRUE	TRUE	Alive	Killed	Killed
97		242	!TRUE	TRUE	Alive	Killed	Killed
98		251	TRUE	!TRUE	Alive	Alive	Killed
104		295	TRUE	!TRUE	Alive	Alive	Killed
106		309	!TRUE	TRUE	Alive	Alive	Alive
107		313	TRUE	!TRUE	Alive	Alive	Alive
110		326	TRUE	!TRUE	Alive	Alive	Killed
111		328	TRUE	!TRUE	Alive	Killed	Killed
112		351	TRUE	!TRUE	Alive	Killed	Killed
174	person.php	136	TRUE	!TRUE	Alive	Killed	Killed
175		140	===	!==	Alive	Killed	Killed
176		141	==	!=	Alive	Killed	Killed
177		142	TRUE	!TRUE	Alive	Killed	Killed
178		150	TRUE	!TRUE	Alive	Killed	Killed
179		154	TRUE	!TRUE	Alive	Killed	Killed
180		158	!TRUE	TRUE	Alive	Killed	Killed
182		162	TRUE	!TRUE	Alive	Alive	Killed
184		172	TRUE	!TRUE	Alive	Killed	Killed
185		180	!TRUE	TRUE	Alive	Killed	Alive
186		181	!TRUE	TRUE	Alive	Killed	Killed
187		182	!TRUE	TRUE	Alive	Killed	Alive
188		183	!TRUE	TRUE	Alive	Killed	Killed

A.3.2 TaskFreak

Mutant Number	Capture/Replay	Random	PBGT
1	Killed	Killed	Killed
2	Killed	Killed	Killed
3	Killed	Alive	Killed
4	Killed	Killed	Killed
5	Killed	Alive	Killed
6	Killed	Alive	Killed

Apendix

11	Killed	Alive	Killed
12	Alive	Alive	Killed
17	Killed	Alive	Killed
23	Killed	Alive	Killed
24	Killed	Alive	Killed
25	Killed	Alive	Killed
32	Alive	Alive	Alive
33	Alive	Alive	Alive
34	Killed	Alive	Killed
35	Killed	Alive	Killed
36	Killed	Alive	Killed
37	Killed	Alive	Killed
38	Alive	Alive	Alive
39	Killed	Killed	Killed
40	Killed	Alive	Alive
41	Killed	Alive	Killed
42	Killed	Alive	Killed
43	Alive	Alive	Alive
45	Killed	Alive	Killed
46	Killed	Alive	Killed
47	Killed	Alive	Killed
48	Killed	Alive	Killed
50	Killed	Alive	Killed
51	Killed	Alive	Alive
53	Alive	Alive	Killed
54	Alive	Alive	Killed
56	Alive	Alive	Alive
67	Killed	Alive	Killed
68	Killed	Alive	Killed
82	Killed	Alive	Killed
83	Killed	Alive	Killed
84	Killed	Alive	Killed
85	Killed	Killed	Killed
86	Killed	Alive	Killed
88	Killed	Alive	Killed
92	Killed	Alive	Alive
93	Alive	Alive	Killed
94	Killed	Alive	Killed
95	Killed	Alive	Killed
96	Killed	Alive	Killed
100	Killed	Alive	Killed
109	Killed	Killed	Killed
113	Killed	Alive	Killed
118	Killed	Alive	Killed
119	Killed	Alive	Killed
120	Killed	Alive	Killed
121	Killed	Alive	Killed
122	Alive	Alive	Alive
123	Killed	Alive	Killed
152	Killed	Alive	Killed
154	Killed	Alive	Killed
156	Killed	Alive	Killed

157	Killed	Alive	Killed
159	Killed	Alive	Killed
160	Killed	Alive	Killed
161	Killed	Alive	Killed
162	Killed	Alive	Killed
163	Killed	Alive	Killed
164	Alive	Alive	Killed
165	Killed	Alive	Killed
170	Killed	Alive	Killed
171	Killed	Alive	Killed
172	Killed	Alive	Killed
173	Killed	Killed	Killed
176	Killed	Killed	Killed
177	Killed	Killed	Killed
178	Killed	Alive	Killed
179	Killed	Alive	Killed
180	Killed	Alive	Killed
181	Killed	Alive	Alive
183	Killed	Alive	Killed

A.3.3 Tudu Lists

Mutant Number	Capture/Replay	Random	PBGT
1	Killed	Alive	Killed
4	Killed	Alive	Killed
5	Killed	Alive	Killed
6	Killed	Alive	Killed
7	Alive	Alive	Alive
8	Killed	Alive	Killed
9	Killed	Alive	Killed
10	Killed	Alive	Killed
11	Killed	Alive	Killed
19	Killed	Alive	Killed
20	Killed	Alive	Killed
21	Killed	Alive	Killed
23	Killed	Alive	Killed
24	Killed	Alive	Killed
25	Killed	Alive	Killed
28	Killed	Alive	Killed
29	Alive	Alive	Alive
30	Killed	Alive	Killed
31	Killed	Alive	Killed
32	Killed	Alive	Killed
33	Alive	Alive	Killed
35	Killed	Alive	Killed
55	Killed	Alive	Killed
56	Alive	Alive	Killed

57	Killed	Alive	Killed
62	Killed	Alive	Killed
63	Killed	Alive	Killed
69	Killed	Killed	Killed