Automated Model-based GUI Test Cases Generation from Use Cases

PH.D. Thesis Proposal
Supervised by Prof. João Pascoal Faria and co-supervised by Prof. Ana Paiva

Department of Informatics Engineering
Engineering Faculty of Porto University
July 2008
Abstract

Mostly of today’s computer users interact with the software through a graphical user interface (GUI). In software engineering, software testing and quality have become a topic of major concern. Software testing is today an important stage in software projects and GUI testing is also crucial to the solution’s quality. Nevertheless, GUI testing is not an easy task, it’s very time consuming and too expensive. Model-based testing is a recent approach that helps to systematize the GUI testing process. However, this technique is not widely applied to GUI Testing and does not support Test Web Applications Testing. Taking advantage of a higher degree of automation, the purpose of this work is to study and develop the generation of Model-based GUI test cases from high-level models of software. This tool will automatize the process of generating GUI Test Cases from semi-formal models like there are UML Use Case or Activity Diagrams. These models can eventually generate executable tests with UML + action language. They also can be enriched with contracts (pre and post-conditions) in order to achieve a higher level of automation in Test Case generation. Success in the resulting tool proposed in the work, would turn GUI Testing more systematic, automated and accessible to a much wide range and variety of users.

Keywords: GUI modelling; GUI testing, model-based Testing, test generation
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 The Proposal (Thesis Definition)</strong></td>
<td>2</td>
</tr>
<tr>
<td>2.1 The Problem</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Thesis Statement</td>
<td>4</td>
</tr>
<tr>
<td><strong>3 State of the Art</strong></td>
<td>5</td>
</tr>
<tr>
<td>3.1 Test Driven Development</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Model-Based Testing</td>
<td>7</td>
</tr>
<tr>
<td>3.3 GUI Modelling</td>
<td>7</td>
</tr>
<tr>
<td>3.3.1 FIT</td>
<td>7</td>
</tr>
<tr>
<td>3.3.2 Use Cases</td>
<td>7</td>
</tr>
<tr>
<td>3.4 Automated Test Case Generation</td>
<td>9</td>
</tr>
<tr>
<td>3.5 Model-Based GUI Testing</td>
<td>9</td>
</tr>
<tr>
<td>3.5.1 Existing tools</td>
<td>10</td>
</tr>
<tr>
<td>3.5.2 Related Work</td>
<td>10</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Nowadays, our society depends ever more on software, and quality is a key to the success of it. In software engineering, software quality has become a topic of major concern. As software is becoming critically important for an organization to be competitive in its business, the requirement that the software is highly supportive for the organization in achieving its goals means that the software should have high utility and user quality.

Software testing is today an important stage in software projects. It is one of the most expensive and time consuming stages that usually stops after using the available resources or even at the middle of the process due to duration of this phase. Manual testing can be described as an action where a programmer initiates each test, interacts with it, interprets, analyzes, and reports the results. Software testing is automated when there is a mechanism for tester-free running of test cases [10]. Testing automation may only include execution automation. It also may include the automation of the generation of test cases and the check of conformity of the results. Automated tests should be accomplished with the least amount of the user involvement.

Test automation is a major challenge for test automation activities. Most of the existing Test automation tools are partially automated and require the user interaction with the tool to complete the tests in different stages of the process.
Chapter 2

The Proposal (Thesis Definition)

2.1 The Problem

Most of today’s computer users interact with the software through a graphical user interface (GUI). GUIs have become more rich, more user interactive and more sophisticated over time. In many applications, one of the major improvements that are suggested with the new releases is to improve the user interface or just a new fresh design for the application or the product. Therefore, GUIs take an important role of interacting with today’s software.

There are different types of testing, like Unit Tests, Integration Tests and GUI Tests. Unit Tests can be very important and useful for software development, but they test isolated subsystems and typically at class level. Integration tests are also very useful and common, but are difficult to set up subsystems in combination.

It’s also well known, that system tests are crucial to the software quality. Nowadays, with such a global market like it’s the market of software solutions, GUI testing is also crucial to commercial solution’s quality.

Nevertheless, GUI testing is not an easy task, it’s very time consuming and too expensive for a massive use by software houses or common developers.

Nowadays, Automated Test Cases Generation is a very important topic of research in GUI Testing. Manual generation can take much time and can cause errors that can reduce the quality and accuracy of the tests.
There are also very few tools available and most of all are not very suitable to the user needs. One of the major problems of these tools, among others [26], is that, these tools still require much manual effort and this testing activity is executed at the end of the development process when the GUI of the software is already constructed and full-integrated. Another important issue of these tools is that most of them require a specialist or trained person to use it properly.

Model Based Testing is an approach that helps to systematize the GUI testing process. However exists a big reluctance of GUI testers and modellers in writing textual formal specifications, they strongly prefer graphical notations with which they are familiar.

2.2 Motivation

There is also very few research on Model-based GUI Testing [14], [17] and the existing research is recent.

Web Applications have a specific GUI, very different from standalone traditional GUI's applications. Therefore, existing GUI testing Techniques cannot be directly applied to Web Applications. Also, there are not many tools to Web GUI Testing. The first tools were mainly used on performance tests, like http loads [11] or server analysis [23].

After, some other tools like Selenium [18], one of the most-used web based testing software tools, started to include featuring testing, user interface and acceptance testing. Actually, there are several sorts of Testing tools on Web Applications. With these tools it is possible to use various methods of Testing, like Unit Testing [21], Capture/Replay Testing [19] and Monkey Testing [16]. At the beginning, one of the major issues, was that all the testing process had to be executed manually. Nowadays, Automated Testing is already used on almost all of these tools, for instance, HttpUnit [22] and Selenium [18].

However, none of the existing tools uses Model-based techniques to automatically generate the test cases and the final GUI. All of them, are non Model-based tools. Tools like UIDE [5] or JANUS [9] are Model-based, but are only used to test stand-alone applications.
2.3 Thesis Statement

After some literature review, study and analysis of related work on this area, we were able to make a thesis statement:

*It is possible to automatically generate test cases from UML Use Cases enriched with contracts (pre and post-conditions) in order to improve the process of building a battery of tests to use on TDD approaches. Improving the process of building tests, in this case, means permit building test cases in visual UML notation, automatically generate concrete test cases (the code of tests) and automatically perform these tests.*
Chapter 3

State of the Art

Traditionally, the testing phase is executed at the end of any software development project. However, it is known [15] that the cost of changing source code or changing the requirements of the software has a much higher cost at the later phases of development. The figure 3.1 presented by Barry Boehm [8] shows how the cost of change grows during the project development.

![Figure 3.1: Traditional cost of change curve](image-url)
3.1 Test Driven Development

Test Driven Development (TDD) is an approach that solves some of the problems existing in traditional software development testing. These problems are, for example, costs of late testing, non-based source code tests or non-automated tests.

On this approach, Beck [6] purposed a development cycle where tests are written first, before the development, or rather, when a new functionality is added to system, the test is written first. Then, the development is made when the test runs and is executed to prove it fails. Then, the method is executed making the unit test succeed. After, the code is refactored to clean and remove duplications. This development cycle has five basic steps:

- **Quickly add a test** - The first step is to write a test case. This test case is focused on a needed functionality for the system.

- **Run the test and see him fail** - The test runs immediately after been written with any development to permit it succeeds. If this new test succeeds with any change on the source code, then, it’s probably an erroneous or unnecessary test.

- **Make little changes on the code** - After the test fails, just little changes are made to test succeeds. The TDD is made with these little developments not connected and few methods, but at the end all together make the whole software solution.

- **Run the test and see him succeed** - On this step, the test runs and the developer has to verify if it succeeds. If it succeeds, the developed code addresses the tested requirements. In other hand, if it fails, the source code needs to be modified and tested again.

- **Refactor the code to clean and remove duplications** - On this last step, the refactoring, the source code is cleaned to remove just duplicated code. No logical modifications should be made on this phase. Finally, after these modifications, all test cases should run once more to test if the refactoring was done correctly, without damaging any functionality of the system.

There are several advantages of the TDD approach of writing tests first and guide the development for testing. Some of these advantages are: Errors are detected
more quickly and can be easily fixed; Quality improvement, the developer has to think in all functionality requirements before writing the code; Test cases can run first and after the source code is written.

3.2 Model-Based Testing

Model-Based Testing (MBT) is a recent approach that helps to systematize the GUI testing process. On this method, test cases are derived in whole or partially from a model that describes some aspects of the system under test.

3.3 GUI Modelling

3.3.1 FIT

Fit is a tool for enhancing collaboration in software development and software testing. Fit helps with the writing of automated acceptance tests. Developed by Mark Cunningham [7], Fit is a simple technology based on HTML tables and has simple usage. This simplicity allows allows clients, testers an programmers to verify what the software does and what it should do.

The test cases are displayed in HTML tables and behind each table is a Java class called fixture. Fit uses JUnit to run the test cases. The contents of the HTML table are took by the fixture and it runs the test cases on the solution being tested.

The testing running process has a very simple layout as can be seen of Figure 3.2. When a test runs, cells are colored to represent the result of each test. If the table cell is colored in green the test succeeded, if the cell is colored in green the test failed. It’s also displayed the expected result and the actual result.

3.3.2 Use Cases

A use case is a scenario that with a visual notation describes the usage of a system by an actor to achieve a specific goal. An actor is a type of user which take part in a sequence of activities with the system . Actors represent people that will use
Figure 3.2: A Fit document showing success (green) and failure (red)

the solution but actors can be computers systems too. Uses cases can be described by normal flow of events and exceptional flow of events - scenarios.

A scenario is a sequential course of events that describe the interactions between an actor and the system. A use case model consists of the collection of all actors and all use cases that describes the whole system. Use cases can be very useful to:

- Capture functional requirements
- Help to develop System Test Cases
- Identify major classes and relationships
- Help the developer to better understand user requirements
- Test case scenarios or Test Scenarios (system, user acceptance and functional)
can be directly derived from use cases (see also limitations with respect to test cases).

Test case scenarios can include parameters or contracts (pre and post-conditions), which can be very useful to improve the process of building a battery of tests.

### 3.4 Automated Test Case Generation

Testing a System during the initial phase of Specification/Design is crucial to the quality and acceptance of the software. The manual generation of Test cases can be extremely time-consuming and expensive. It also can be erroneous that can endanger the quality of the final solution.

Automate Test Case Generation is the process of build or generate Test Cases automatically from different approaches, mainly formal models. However, research on this field is still on a early stage of development. Nevertheless, there are already some tools UML-Based, specially for unit and system testing like IBM Rational Robot, Mercury Win Runner, Mercury Quick Test, Empirix and Segue, that can produce some satisfying results. More recently, model-based approaches have been under a deeper research, specially due to their potential to automate test generation [17].

### 3.5 Model-Based GUI Testing

Model-Based Testing applied to GUI is one of the most recent fields of research in this area. However, due to an increasing interest in model-based approaches, exist some tools that are based on this methods. As mentioned above, currently it is extremely time-consuming and expensive, and exist few tools that generate test cases and evaluate the GUI testing Most of the tools that automate GUI testing do not address these two aspects [4].
3.5.1 Existing tools

Existing Model-Based GUI Testing tools [1][13] are still very simple. Mainly, these tools use Capture/Replay and Random input testing methods [24][20]. They ease the construction of test cases but still require much manual effort. Also, the testing task is done at the end of the development process when the changes in the solution have a much higher cost.

3.5.2 Related Work

There exist few research on the field of Automated Model-Based GUI Test Cases Generation. Atif M. Memon was the first to present research work on this field. On his first works [14][13][3] presented some techniques from Artificial Intelligence to automatically generate GUI tests and a solution for regression testing. More recently [2], developed an approach to reverse engineer a model directly from an executable GUI.

Recently, Ana Paiva [17] proposal introduces more systematization and automation into the GUI testing process with automatically generated test cases. Thus, this purpose represents a significant improvement over the current GUI testing approaches based on Capture/Replay tools, since they only automate the execution and recording of the test cases.

Nevertheless, reluctance of modellers and testers to write formal specifications and the weak support of coverage criteria for GUI testing are still some of the problems commonly addressed to this approach.
Chapter 4

Objectives

The purpose of our work has two distinct phases. In the first phase, we plan to study and propose a solution for the problem here identified. In the second phase, a tool will be developed to address and confirm the proposed solution. Therefore, at the end of this project we aim to reach the following deliverables:

• Propose a solution to introduce more systematization and automation into the GUI testing process with a Model-Based approach with Automated Test Cases Generation.

• Study how Model-Based GUI Testing techniques can be applied to Web Applications. This type of approach was never been studied and applied before on Web Applications.

• Develop a tool to automatically generate test cases from high level models of the software (GUI) under test that addresses the above mentioned desirable solution. This tool will automatize the process of generating GUI Test Cases from semi-formal models like there are UML Use Case or Activity Diagrams. These models can eventually generate executable tests with UML + action language. They also can be enriched with contracts (pre and post-conditions) in order to achieve a higher level of automation in Test Case generation.

• Apply the developed tool to test Web Applications and improve the Testing of such specific type of applications. Current solutions are mainly Capture/Replay and Random Input tools. On this approach, the goal is to test the all solution and it functionalities.
Chapter 5

Methods

On this project, we can divide the work in four distinct phases: Information Gathering; Thesis Definition; Approach’s Development; Approach’s Evaluation.

Zelkowitz and Wallace [27] proposed some engineering validating methods for validating technology. They are categorized in three categories: Observational, Historical and Controlled. Here are the methods we expect to use in each of the phases of our work:

- **Information Gathering Phase** - Literature Search, Legacy and Project Monitoring
- **Thesis Definition** - Literature Search, Assertion and Project Monitoring
- **Approach’s Development** - Replicated, Synthetic, Dynamic Analysis
- **Approach’s Evaluation** - Synthetic, Dynamic Analysis and Simulation

On this proposal, we do not yet have a high-level of detail in order of which are the methods that will be used. However, we expect to use mainly the methods above mentioned.

As mentioned before, during this research work, a tool will be developed to automatically generate test cases from high level models of the software (GUI) under test. On the development of this system distinct methods will be used like transforming UML visual notations and its conversion to a interchange format...
like XMI [25] using some techniques via XLST like Kovse described [12]. The approach promotes model reuse, speeds up the modeling process and can be used to assure that only predefined semantics (as specialized by an agent) is included in the transformed model.

Other method that will be used is using techniques to generate the test cases depending in the type of the UML Visual notation. These notations can be activity diagrams, state charts, etc. Use cases will be used as parametrized test scenarios.

5.1 Evaluation

The basic evaluation on this project will be done with the developed tool that will allow to automatically generate the test cases. This stage is to verify if the tool deployed by our proposal produce valuable results. Therefore, we will have some case studies with high level models with the main functionalities of the program.

There are several models and methods for validating technology as Zelkowitz and Wallace [27] proposed and is mentioned above.

On this project we expect to use some observational methods like case studies that can monitor project in depth and assertion with some ad hoc validation techniques. We also intend to use controlled methods like simulation and syntethic methods that can be automated and are specially indicated to validation of developed tools.

During the development of the project and mainly after the development of the tool a deeper research will be performed to find the most appropriate methods to evaluate the proposed solution.
Chapter 6

Research Plan

Following is the proposed time line for the project.
The present work plan was defined considering 3 years to complete the project including the writing and thesis dissertation defense. The project starts in September 2008 and finishes on the end of August 2011.

• From September 2008 to March 2009 - Literature study and review - The research and literature study already has started and will be the research work for the State of the Art writing.

• From April 2009 to July 2009 - Survey of Requirements - On this phase, that will take four months to be completed, there will be done a survey of all necessary requirements to the Project execution.

• From August 2009 to June 2010 - Build Test Case Generation Tool - Development of the Test Case Generation tool that will be application of the proposed thesis.

• From July 2010 to November 2010 - Assembly/Testing - Deploy of the developed tool and it will be ensured that all tool’s components function properly with the system requirements when assembled.

• From December 2010 to April 2011 - Evaluation Phase - On this phase, it the work will be evaluated to verify and validate all requirements and goals of the project.
• **From July 2010 to August 2011** - Thesis Writing - Some of this work will be done during the Literature review. Final phase of the project, where will the dissertation thesis will be written, with the description of the project, state of the art, methodologies applied, tool development and obtained results.

The average of working hours for this project will be around 20 hours weekly. During the phase of Build Test Cases Generation Tool and Thesis Writing the working hours can go up to 30 hours week.

![Figure 6.1: Work Plan of the Project](image)

### 6.1 Contingency Plan

This project is planned for 3 years. However, the author is not full dedicated to the project due to his professional activity. Therefore, the work is divided in one main objective and several sub-objectives. These objectives will be better defined after the Survey of Requirements, the second phase of the work. Some of the sub-objectives may not be totally satisfied if their realization delay the achievement and execution of the main objective.

The main goal of this project is to develop a prototype of an Automated Model-Based GUI Test Cases Generation Tool from UML Use Cases. It is not a main task of this prototype, demonstrate and support all the functionalities that usually testing tools have. Nevertheless, all the main functionalities identified will be included in the prototype.
Chapter 7

Final considerations

This report describes a proposal of a PH.D. work where we were already addressed the problems that will subject of study. Also, this work here presented allows us to propose some possible solutions to the problems addressed.

Thus, we present a work plan that we think is adequate to the estimated work and to the dedication of the author to this project. Nevertheless, we also present a contingency plan to prevent some delays or possible setbacks during the development of the work.

The project here proposed is not just the development of a tool that can automatically generate test cases from a high level model of a GUI. Success in the work proposed in this document will make Software Testing accessible to a wide range and variety of users. This will enable users with no skills in this area to test complex solutions without the need of a specialist and therefore making such activity more autonomous and more common in software development.

This work also intends to fill the gap between abstract and concrete test cases offering the possibility to work with such high level and widespread notations like UML Models. The UML diagrams are designed in such a way that they can be used for strong communication. In addition to communication between the development team, applications designed with UML will more efficient for the end user.

In other hand, applying these methods to Web Applications will permit to test these applications with such specific GUI’s. Using techniques that now are not
common on these applications, like creating Model-based GUI Test Cases and applying the tool above-mentioned will help to enhance the web development and specially Web Applications Testing.

Success in the work proposed in this document would turn GUI Testing more systematic, automated and accessible to a much wide range and variety of users. We also expect to contribute to the progress of science, specially in the field of Software Quality and Testing.
Bibliography


