POLITECNICO DI TORINO

Master Degree Thesis

Evaluation of the maintenance required by web application test suites

Supervisors
prof. Morisio Maurizio
prof. Riccardo Coppola

Candidate
QI SHUANG
Student number: s233329

Academic Year: 2019-2020

Abstract

Context: With the development of the Internet, the web application occupied a large proportion in the market, every industry needs to have its website, to provide web service. So the web application should be strong enough for use, web application testing can test the application and find the potentiality problems, and shows out how to maintain the web application when it does the evolution. That's why web application testing becomes so important in the latest years, which requires an important effort from developers.

Goal: The objective of this thesis is to understand the development of web applications and four testing techniques for the web application, compare their differences, analyze their advantages and disadvantages. Make the experiment with an existing open-source web application, define some metrics for evaluating the maintenance effort for the application evolution.

Method: Firstly, it will introduce the main research questions of definition Of Web Applications, High-level definition of Testing, Web Applications Testing introduction, Web Application Testing Techniques, Maintenance, and fragility of the web application. Secondly, Do an empirical experiment on open-source web application projects(dolibarr), to analyze the testing suits of web application and define the metrics for analysis. Lastly, Giving the results and analysis by applying testing tools on a web application.

Results: The result of the review shows that many development frameworks and testing tools have been released during the last years, Selenium WebDriver is excellent used testing tool for the web application. Throughout the exploratory study, it has been found that by using Selenium WebDriver testing, by defining some metrics, it will show the maintenance and fragility between web application evolution.

Conclusion: The web application can be tested by different techniques, all of them expose issues that are discussed in the present thesis. Some native testing tools can be leveraged, these testing tools, existing some advantages and disadvantages, web application testing is a complex issue. Selenium WebDriver is powerful for web application testing. It gives out maintenance and fragility between web application evolution. The results of the thesis show that testing technology should grow with the development of web applications for developers to do better testing work for the web application.

Acknowledgements

The candidate warmly thank her parents (father: QI SHI YU and mother: YANG ZHAO XIA)and brother(QI DE TING), for all love and their encouragement, moral support, personal attention, and care. They encourage her all time and support her when she feeling lost and depressed, takes care of her, and loves her without any conditions.

Express candidate sincere gratitude to Asst. Prof. Morisio Maurizio and Asst.Riccardo Coppola, for allowing her to conduct this research under their auspices. She is especially grateful for their confidence and the freedom they gave her to do this work also for the confidential information they have kindly provided by her, and for the useful discussions they have allowed candidates to improve her thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rapid development of network technology, especially the popularity of the Web and its applications, various types of Web-based applications have become the focus of software development due to their convenience, speed, and ease of operation.

The Web is taking the world by storm with its breadth, interactivity, and ease of use, and the number of web pages is growing rapidly by geometric orders of magnitude.

Being able to attract as many users as possible and paying attention to it for a long time is the main goal of the website, and it is also the main indicator to measure the success of the website. So testing becomes an important part of the application development process.

Web-based systems tend to evolve rapidly and undergo frequent modifications, due to new technological and commercial opportunities, as well as feedback from the users. For such reasons iterative development process models, based on the notion of rapid prototyping and continuous change, seem to fit the conditions in which Web sites are produced and maintained.

As web applications become more and more widely used, the requirements for performance testing are also increasing. However, as web programs integrate a large number of new technologies, such as HTML, Java, JavaScript, VBScript, etc., they also rely on many other factors, such as Link, Database, Network, etc., make web application testing very complex.

" Modern Web applications are sophisticated, interactive programs with complex GUIs and numerous back-end software components that are integrated in novel and interesting ways."[20]

Web testing is part of software testing. It is a type of testing for web applications. Because web applications are directly related to users and usually

need to withstand a large number of operations for a long time, the functions and performance of web projects must be reliably verified.

Through testing, it can find as many errors as possible in the browser and server programs and correct them in time to ensure the quality of the application. Because the Web is distributed, heterogeneous, concurrent, and platform-independent, its tests are much more complicated than ordinary programs.

It can be used different testing techniques to test the software and compare the advantages or disadvantages of different testing techniques. Use Selenium web driver to test dolibarr, compare the differences between different versions of dolibarr, and calculate some data. To see how the web application testing is hampered by the fragility issues.

"For the purposes, it is possible to define a GUI test case as fragile if it requires interventions when the application evolves (i.e., between subsequent releases) due to modifications applied to the application under test (AUT). "[6]

As system-level tests, Web application test cases are influenced by different functionalities of the application also for even small interventions in the appearance, definition, and arrangement of the GUI presented to the user.

In the work, it defines 15 test cases for testing dolibarr, for 5 versions of this application. When updating the application from a lower version to a higher version, it needs to modify the test cases to make the testing work.

It will be recorded the changes due to the evolution, and define some metrics. It is measured the relevance of testing code concerning the total production code for each project in terms of quantitative comparisons of the respective amount of lines of code.

To estimate the fragility issue, It is possible to define a set of metrics that can be obtained for each project by automated inspection of the source code. Thus, "we can give a characterization and a quantification of the average fragility occurrence for each of the testing tools considered. "[6]

It did a proper quantitative analysis for the test suites for the application. also, it is possible to define these metrics, they can be a standard to evaluate the maintenance effort for updating the application.

Based on this fragility evaluation, it can be possible in the future to define a taxonomy of fragility causes for scripted Web application testing and to give more in-depth actionable guidelines for developers to circumvent some of them.

The remainder of this paper is organized as follows. Part 2, give background information for web application and the high level of testing; The

definition of web application testing; Introduce 4 different techniques for web testing; Maintenance and fragility. Part 3, the experiment by using selenium WebDirver; introduction of dolibarr; define test cases; define metrics; how to compute the metrics; the results from the work. Part 4, Conclusion, and future work.

Chapter 2

Background

2.1 Introduction to Web Applications

2.1.1 Definition Of Web Applications

A web application is an application that operates on the Internet or an intranet using a web browser. An application is written in a web language (such as HTML, JavaScript, Java, or other programming languages) that requires a browser to run."Web applications employ several new languages, technologies, and programming models, and are used to implement highly interactive applications that have very high-quality requirements."[20] It can be run directly on various computer platforms without the need for pre-installation or regular upgrades.

2.1.2 Working Principle

The application has two modes, one is the client/server program, which means that such programs generally run independently. Another is a browser-side / server-side application. This type of application generally runs with the help of a browser such as IE. Web applications are generally in browser-side / server-side application mode. Web applications are first "applications", and are not fundamentally different from programs written in standard programming languages such as C, C ++, and so on. However, web applications have their unique features, that is, they are based on the web, rather than running traditionally. In other words, it is the product of a typical browser/server architecture.

A web application is composed of various web components that accomplish

specific tasks and expose services to the outside world through the web. In practical applications, web applications are composed of multiple servlets, JSP pages, HTML files, and image files. All these components coordinate with each other to provide users with a complete set of services.

2.2 High-level definition of Testing

2.2.1 Definition Of Testing

Software Testing describes a process used to facilitate the verification of the correctness, integrity, security, and quality of software.[11] In other words, software testing is a review or comparison process between actual output and expected output. The classic definition of software testing is the process of operating a program under specified conditions to find program errors, measure software quality, and evaluate whether it can meet design requirements.

Organizations that are developing software solution are faced with the difficult choice of picking the right software development life cycle (SDLC).[10] The SDLC defines the steps involved in the development of software at each phase and covers the plan for building, deploying, and maintaining it.

The waterfall model is a sequential design process, which is usually used in the software development process.[10]

In this process, the progress is considered to flow steadily downward (such as a waterfall) through various stages. The V model represents an extended software development process that can be regarded as a waterfall model.[10]

In software development,the V-model, also known as Verication and Validation Model, is a Software Development Life Cycle (SDLC) model where instead of moving down in a linear way (as in the waterfall model) the process steps are bent upward having the development of each step directly associated with its testing phase.[9]

There is the V-Model in figure 2.1:

The verification phase begins by collecting system requirements by analyzing user requirements. In this step, a requirements document will be generated. Later, the system is designed by studying the generated documents. After the design is completed, coding guidelines and standards will be used to transform the coding stage. The final stage, V, is the verification stage. In this step, the test designed with the counter part of the verification step is performed in the reverse order. At this stage, we can determine four main test levels:

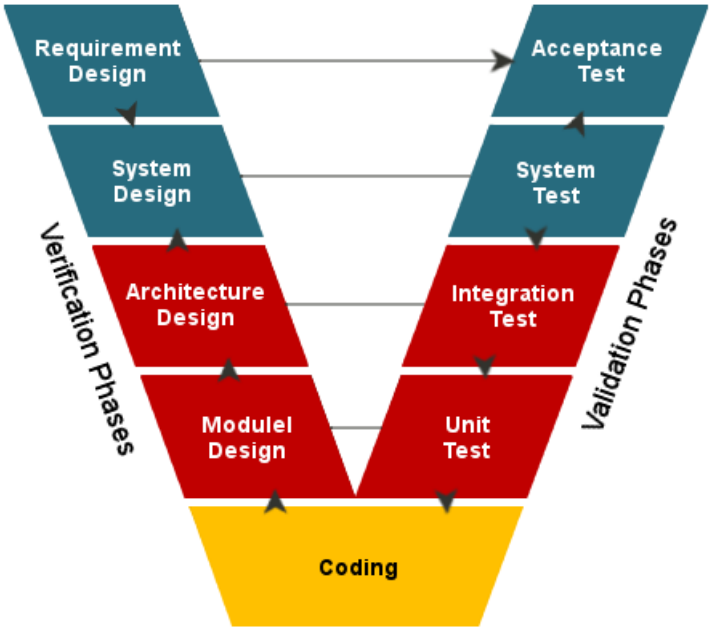


Figure 2.1: V-Model.

1. Unit testing
2. Component testing
3. System integration testing
4. Acceptance testing

2.2.2 The Objective Of Testing

Software testing is the process of executing a program to find errors. Testing is to prove that the program is wrong, not to prove that the program is error-free. (Finding errors is not the sole purpose) A good test case is that it finds errors that have not been discovered so far. A successful test is a test that has found no errors so far.

Testing is not just about finding errors. By analyzing the causes of errors and the distribution characteristics of errors. It can help project managers find defects in the software processes currently used for improvement. "Based

on the analysis for Web application characters and traditional software testing process, the process for Web application testing is modeled, which describes a series of testing flows such as the testing requirement analysis, test cases generation and selection, testing execution, and testing results analysis and measurement." [21]At the same time, analysis can also help to design targeted detection methods to improve the effectiveness of the test. A test without errors is also valuable, and a complete test is one way to assess the quality of the test.

2.3 Web Application Testing

Speaking of web application testing, Web testers must deal with shorter release cycles, and testers and test managers face a shift from testing traditional Client/Server structures and framework environments to testing rapidly changing Web applications. There are many kinds of tests for Web applications, such as the functionality testing, performance testing, safety testing, usability testing and compatibility testing; and the testing methods are divided into the white box testing, black box testing and grey box testing.[21]

2.3.1 Functional Testing

Link test

Link is a main feature of the Web application system, it is the main way of switching between pages and guiding users to some pages that do not know the address. Link testing can be divided into three areas.

Firstly, test whether all the links are linked to the linked page as indicated.

Secondly, test whether the linked page exists.

Finally, ensure that there are no orphaned pages on the web application. The link test must be completed during the integration-test phase, that is, the link test is performed after all pages of the entire web application system have been developed.

Form test

When the user submits information to the Web application system administrator, it needs to use form operations, such as user registration, login, information submission, and so on. In this case, it must test the integrity of the commit operation to verify the correctness of the information submitted

to the server. If the default value is used, also verify the correctness of the default value.

Cookies Tests

Cookies are usually used to store user information and user operations in an application. When a user accesses an application system using cookies, the web server will send information about the user and store the information in the form of cookies. If the web application uses cookies, it must check if the cookies work.

Design Language Testing Differences

In the Web design language version can cause serious problems on the client or server side, such as which version of HTML to use. This problem is especially important when developers are not working together when developing in a distributed environment.

Database Testing

In the Web application technology, the database plays an important role, the database provides space for the management, operation, query and implementation of the user's request for data storage. Two types of errors may occur, namely data consistency errors and output errors. Data consistency errors are mainly caused by incorrect form information submitted by users, and output errors are mainly caused by network speed or programming problems.

2.3.2 Performance Test

Connection Speed Test

The speed at which users connect to a web application varies depending on how they are connected. They may be dial-up or broadband. If the web system responds too long (for example, more than 5 seconds), the user will leave without waiting patiently. Besides, some pages have a timeout limit. If the response speed is too slow, the user may not have time to browse the content, and then need to re-login.

Load Test

The load test is to measure the performance of the Web system at a certain load level to ensure that the Web system can work within the requirements. The load level can be the number of users accessing the Web system at the same time, or the number of online data processing.

Stress Test

Stress testing is the test system's limitations and failure recovery capabilities, that is, testing whether the web application system will crash and under what circumstances it will crash. Hackers often provide erroneous data loads until the web application crashes and then gain access when the system is restarted.

2.3.3 Usability Test

Navigation Test

Navigation describes how the user operates within a page, between different user interface controls, such as buttons, dialogs, lists, and windows, or between different connected pages.

The users would quickly scan a Web application system to see if there is information that meets their needs, and if not, they will leave quickly. so the web application navigation help should be as accurate as possible.

Make sure that the user knows intuitively whether there is content in the web application and where the content is. It is necessary to start testing the user navigation function, so that the end user can participate in this test, the effect will be more obvious.

Graphics Test

In the Web application system, the appropriate pictures and animations can not only play the role of advertising, but also can beautify the page. A web application's graphics can include images, animations, borders, colors, fonts, backgrounds, buttons, and more. The contents of the graphical test are:

1. To ensure that the graphics have a clear purpose, the image size of the web application system should be as small as possible, and to clearly explain something, it is generally linked to a specific page.

2. Verify that the styles of all page fonts are consistent.
3. The background color should match the font color and foreground color.
4. The size and quality of the picture is also a very important factor, generally using JPG or GIF compression.

Content Testing

Content testing is used to verify the correctness, accuracy, and relevance of the information provided by Web applications. The correctness of information refers to whether the information is reliable or misrepresented. the accuracy of the information refers to whether there are grammar or spelling mistakes. This type of testing is usually done using some word processing software, such as the "Pinyin and Grammar Check" feature of Microsoft Word; the relevance of the information refers to whether the current page can find a list or entry of information related to the currently viewed information, that is, The so-called "related article list" in a general Web site.

The Overall Interface Test

The overall interface refers to the page structure design of the entire web application system, which is a sense of overallity for the user. The general web application system takes the form of a questionnaire on the homepage to get feedback from the end user.

2.3.4 Client Compatibility Testing

Platform Testing

There are many different operating system types on the market, the most common ones are Windows, Unix, Macintosh, Linux, etc. Which operating system the end user of the web application uses depends on the configuration of the user system. before the release of the Web system, it is necessary to perform compatibility testing on the Web system under various operating systems.

Browser Testing

The browser is the core component of the Web client. Browsers from different vendors have different support for Java, JavaScript, ActiveX, plug-ins or different HTML specifications. In addition, the framework and hierarchy styles

are displayed differently in different browsers, or even not at all. Different browsers have different settings for security and Java.

2.3.5 Safety Test

The security test areas of the Web application system mainly include:

- The current web application system basically adopts the method of registering first and then logging in. Therefore, it must test valid and invalid usernames and passwords, be aware of the case sensitivity, how many times it can try, whether it can browse a page without logging in, etc.

- Whether the web application system has a timeout limit, that is to say, after the user logs in after a certain period of time (for example, 15 minutes), no page is clicked, and it is necessary to re-login to be used normally.

2.4 Qualitative Evaluation of Web Application Testing Techniques

2.4.1 Model based Testing

Model-based testing (MBT) is an automation of the black-box testing technique. Its main dissimilarity from the conventional black-box testing methods is that the test cases are automatically generated via software tools that exploit the expected behavioral models of the software under the test (SUT). [3] It is a lightweight, formal method of validating software systems.

"Model-based testing (MBT) is a variant of testing that relies on explicit behavior models that encode the intended behaviors of a SUT and/or the behavior of its environment".[22]

First of all, model-based testing of the software system to be tested (often referred to as System Under Test, SUT for formal modeling), to design a machine-readable model; Second, different from other formal methods, model-based testing is not designed to keep the software system under test consistent with specifications in all possible situations, but rather to systematically generate a set of test cases from the model, using this set of test cases to test the software system under test. There is sufficient evidence that the behavior of the system under test is consistent with the model expectations.

First, It needs to create a machine-readable model that expresses all possible behaviors expressed by the requirements. This step is done manually and is the most productive step in the entire process. The key to model design work is the correct abstraction.

A modeler should focus on one aspect of the system to be tested without having to care about the rest of the system. Different parts can be covered by different models, but each model ensures that it is on a clear abstraction level.

Although the workload of creating a model is large, the rewards are huge. By translating informal requirements into formal models, it will be able to easily identify missing parts of the requirements, and it can get feedback on the requirements by analyzing the models.

In terms of MBT, the necessity of validating the model implies that the model must be simpler (more abstract) than the SUT, or at least easier to check, modify, and maintain. Otherwise, the efforts of validating the model would equal the efforts of validating the SUT.[22] And then it is possible to use that model to derive or generate test cases. These test cases provide a test sequence to control the system under test while observing the return value of the system to be tested, and comparing it with the expected value, and then making a decision whether the test passed or failed.

Test cases can be executed repeatedly to complete one MBT iteration. During each MBT iteration, models are refined and enhanced, new models are created to respond to new requirements and requirement changes. As the resulting model grows in size and complexity as more functionalities are added to the application.

The advantages of MBT

The approach provides automated support for fine-grained control of workload characteristics. For example, the approach would make it easy to create controlled workloads to study how varying the characteristics of a particular workload attribute impacts system performance. [1]

It can help reduce the escaped defects compared with the traditional testing approach, which means it can catch requirement defects much earlier in the development process. This is because requirements defects are often detected during the modeling process while the software is still being designed and developed.

The disadvantages of MBT

The main liability of MBT is to know how to model SUT. [4] For producing the prototype or model, the testers need to have essential abilities, also it is quite complex to create the model, it takes a long time to do this task. a small increase in the complexity of the web application would cause an explosion in the number of additional states and transitions.[4]

The kind of models are used by model based testing tools

There are three models that can be used for testing.

1. Finite State Machines

   A finite state machine is a tool for modeling object behavior. Its function is mainly to describe the sequence of states that an object has experienced during its life cycle, and how to respond to various events from the outside world.

2. State Charts

   A state charts is a dynamic behavior that describes an entity's event-based response, showing how the entity reacts to different events based on the current state.

3. Unified Modeling Language (UML)

   UML is a standardized modeling language consisting of a set of diagrams UML has notations such as: Activities,Actors,Business Process,Components,Programming language.

Existing tools using this technique

These years, many MBT tools are developed and used widely in the testing area.

1. Conformiq Creator(Activity Diagrams, DSL).
2. GraphWalker(FMS).
3. MaTeLo(Markov chains).
4. TestOptimal(FSM).

TestOptimal will be introduced.

2.4.2 TestOptimal

TestOptimal is a model-based (MBT) test design and test automation toolset that provides an integrated solution from modeling to test case generation, test automation and test execution, which brings flexibility and efficiency, improve test procedures, and shorten test cycles.[18]

The model has been automated for online model-based testing, which means when the model executes it generates test cases at the same time executes them in real-time.

With TestOptimal, creating a model is quite easy. It's simple as right mouse clicking and right mouse drag and drop. To automate the model, just writing a few lines of MScript for each transition. MScript is essentially an XML script that is very simple and easy to learn.

The general steps of development process using TestOptimal is :

1. Modeling
2. Test Generation
3. Automation
4. Execution
5. Analysis

then come back to modeling, that's why MBT process is an interactive process.

Model used by TestOptimal

The model is described as a state diagram, also known as finite state machine diagram(FSM)

Example of model extraction from Wikipedia with Test Optimal

First, it is possible to install testOptimal, also it needs use Firefox browser with WebMBT builder Add-On installed, after IDE is opened, here it will show "FileList", inside there is a "DemoModel" list, and it shows some demo examples.

So now It starts to create model for Wikipedia, From "File" button, it will show "New Model", click it , then there is the page in figure 2.2:

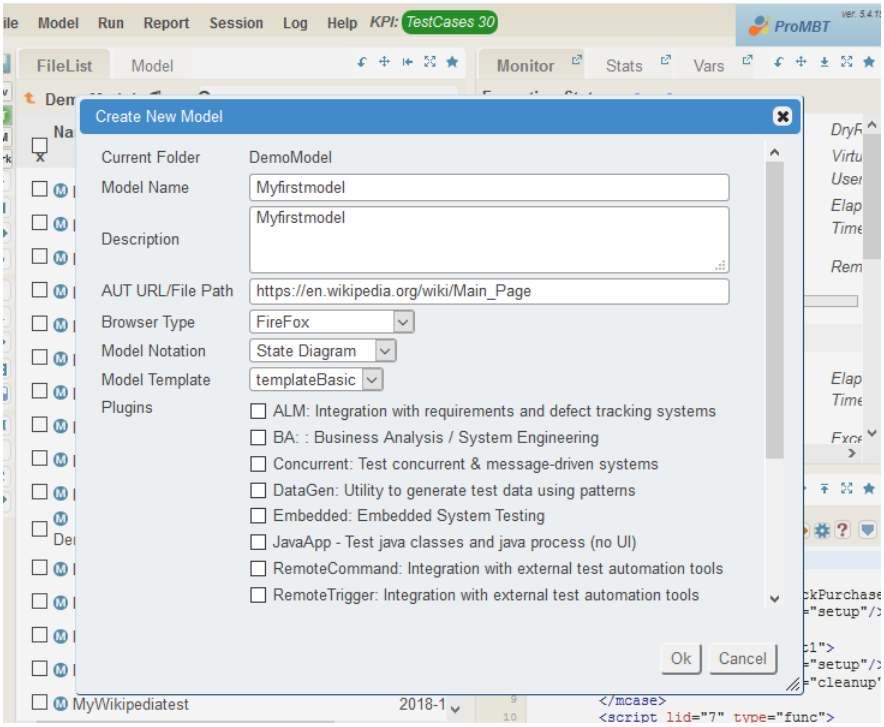


Figure 2.2: Create New Model by using TestOptimal.

Put on “Model Name” the model name, write some words in “Description”, put on “AUT URL/File Path” the URL of the web application, in themodel, it is the main page of Wikipedia. Then click “OK”. Now there is an empty model.

Click “Model”- “WebMBT Builder”, then it will be in the main page of Wikipedia, for creating the initial state of the model, right-click it will show “TestOptimal MBT Builder ”, inside of this there are many choices like “Create State”, “Create Transition”, “MScript-click ” and so on, so choose “Create State”, then come back to the IDE page it will show a new state will be created, then it creates other states like before.

Now creating transitions between different pages, selecting the main page state as the current state, then it will go to Wikipedia main page, right-click, select “Create Transition”, so in the IDE it can be chosen the “To State” the destination page, so here chooses “ Portal: Content ”, it is possible to define Trans ID. It is possible to create other transitions in the same way.

And also define “Final state” of model. Finally it shows model in figure 2.3:

In this model, it just tests some links of Wikipedia pages, There are not

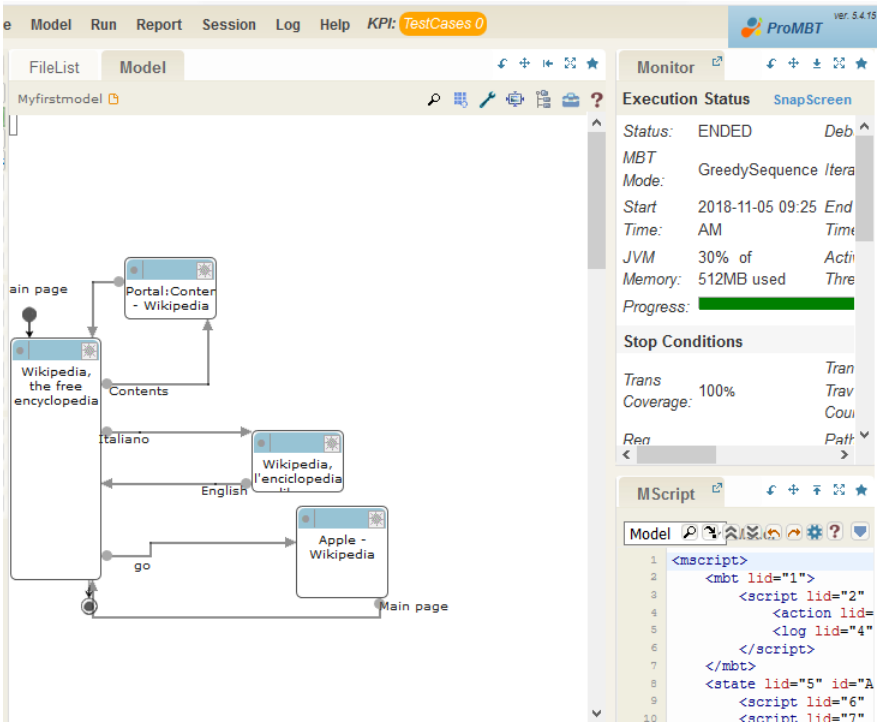


Figure 2.3: The simple Model for Wikipedia by using TestOptimal.

complex operations. So now it can start to run this model, there are three ways to run the model: Run, Debug, Animate/Play. So it is possible to try " Animate/Play", then it will show the model run with animation.

1. Check model coverage in figure 2.4.

   Green means covered.

2. View test sequence generated in figure 2.5.

   This is traversal graph to visualize test cases on the model, failures will be highlighted with red color.

3. View test cases in MSC graph in figure 2.6.

   MSC gives a better visualization of test cases. Failures will be highlighted.

Come back to IDE, It can be seen in the right side there is "MScript", double click it , it will show the full code of the model,this code is auto generated MScript from WebMBT Builder, MScript is organized by triggers for states and transitions, it can be edited as well. Until now the steps of creating the model based testing of Wikipedia by using TestOptimal finished.

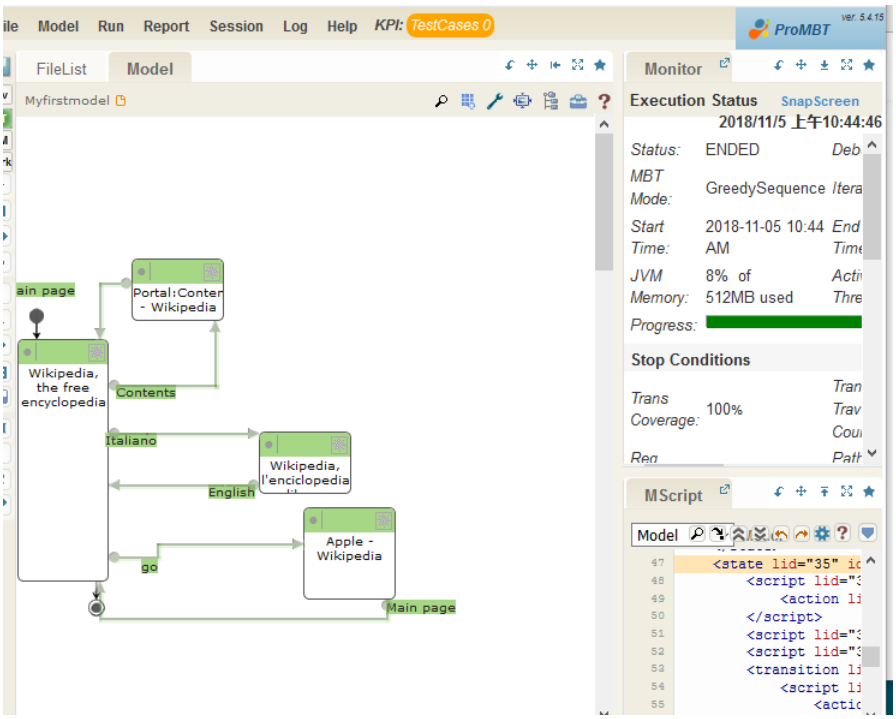


Figure 2.4: Model coverage by using TestOptimal.

2.4.3 Script-based Testing

When talking about Scripted based Testing, it means the testing technique that uses testing scripts to test the system. What is testing scripts? It generally refers to a series of instructions for a particular test, which can be executed by automated test tools.

To improve the maintainability and reusability of testing scripts, they must be built before the test scripts are executed.[28] A testing script is a computer-readable instruction that automatically executes a test process (or part of a test process).

Testing scripts can be created (recorded) or automatically generated using testing automated functional GUI tool(such as HP QuickTest Professional, Borland SilkTest, IBM TPNS and Rational Robot), or programmed in a programming language(such as C++, Tcl, Expect, Java, PHP, Perl, Powershell, Python, or Ruby).

Test engineers write programs using scripting languages (e.g., VBScript), and these programs (test scripts) mimic users by performing actions on GUI objects using underlying testing frameworks.[29] The test scripting language

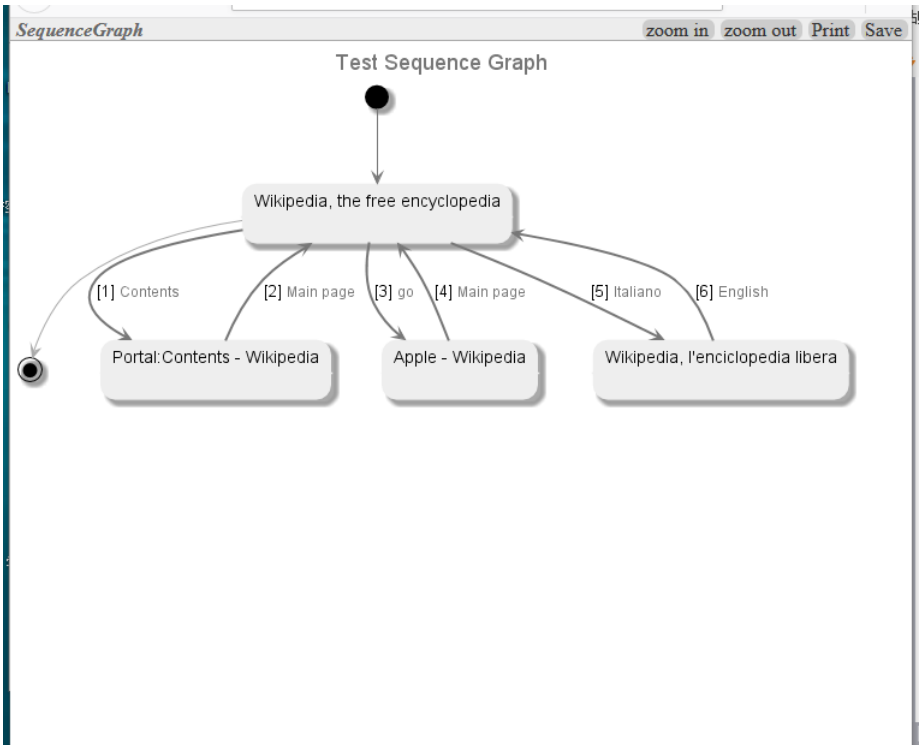


Figure 2.5: Test sequence generated.

is a kind of scripting language. It is precisely a branch of the scripting language in the testing field, which is the basis of the automated software testing design. To understand the testing scripting language, it should understand the scripting language.

The scripting language is a programming language that is interpreted by interpreting during execution. For example, various shells of common Perl, python, PHP, TCL, guile, ruby, and UNIX systems are scripting languages. The efficiency is inferior to the program that is executed after compilation, such as programs written in C, C++, Java, Pascal, etc.

The advantages of Scripted based Testing

Cases require non-trivial programming skills, the involved effort being comparable to that required for normal code development. [2] The scripting language syntax is simple and flexible, and it is not very relevant to efficiency. It is convenient to use scripts. Especially for the processing of multiple files, data flow and processing are more complicated.

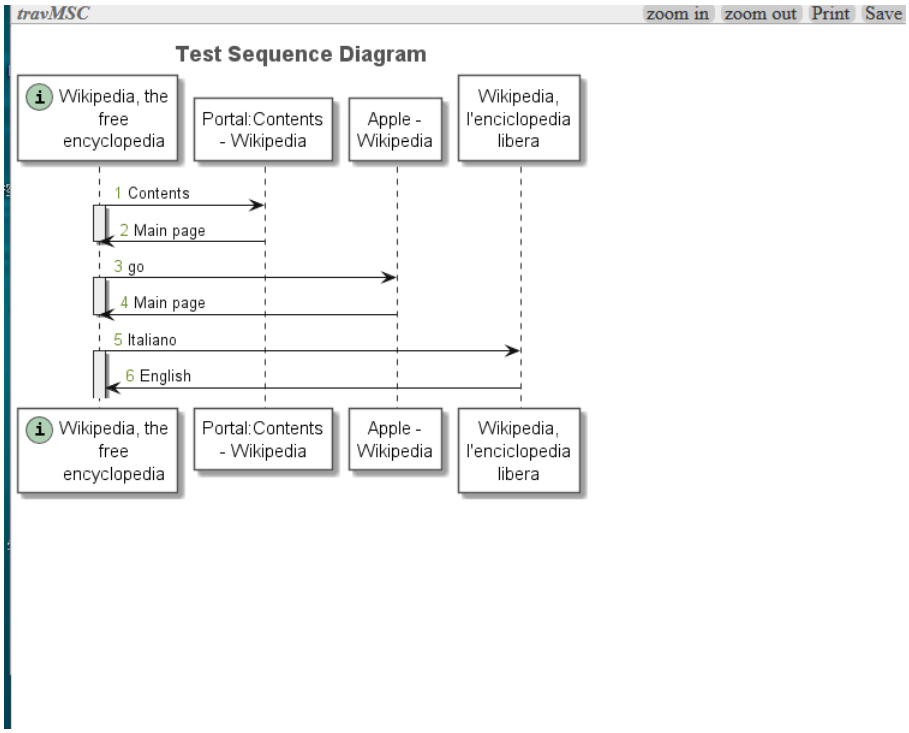


Figure 2.6: MSC graph.

The disadvantage of Scripted based Testing

All benefits of modular programming can be brought also to the test cases, such as parametric and conditional execution, reuse of common functionalities across test cases, robust mechanisms to reference the elements on a web page. [2] It lacks strict compilation process, variable definitions, function definitions, etc. So it is more likely to occur problems which will trouble the testers. The program code lines, functions, code segments, and scripts can be controlled at different granularities. The test is more thorough.

Existing tools using this technique

1. Selenium (Ruby, Java, NodeJS, PHP, Perl, Python, Groovy).
2. Sahi(Sahi script).
3. Maveryx(Java).
4. Ranorex Studio(VB.NET).
5. Selenium WebDriver.

Selenium WebDriver will be introduced.

Selenium WebDriver

Selenium WebDriver is a tool that supports browser automation. It includes a set of libraries and "drivers" for different languages to automate the actions on the browser."Selenium webdriver directly communicate with the browser".[18]

Selenium webdriver supports multiple web browsers and also support for Ajax applications. "The main goal of the selenium webdriver is to improve support for modern web application testing problems." [18] Selenium webdriver supports multiple languages to write the test scripts.

It is a programming interface to create and execute test cases by using Elements locators/Object locators/WebDriver methods. Selenium WebDriver interacts with browser directly, each browser has its own driver on which the application runs, Selenium WebDriver makes direct calls to the browser.

Speaking of WebDriver, it provides libraries for many languages:

- Java
- Ruby
- JavaScript
- Python
- PHP
- Perl

and other languages. This allows developers to use WebDriver to use their most familiar language without having to learn WebDriver's unique scripting language. Using WebDriver is like using a third-party library in the project.

WebDriver uses a "driver" to control different browsers. Currently, supported browsers include Firefox, Chrome, Safari, and IE. Microsoft is developing a new driver for Microsoft Edge. Firefox drivers are built-in, so Firefox is the first choice for automated testing of most projects.

What does WebDriver do?

WebDriver automates the browser. WebDriver can open the URL to interact with the rendered page:

1. Create a new browser instance
2. Open a URL in the browser
3. Click on the link on the page
4. Enter information in the field
5. Execute JavaScript on the page

The advantages of Selenium WebDriver

Support most programming languages, browsers, and operating systems. Overcomes limitations of Selenium like file upload, download, pop-ups. WebDriver's API is simpler than RC's API, it does not contain redundant and confusing commands. Supporting Batch testing, Cross browser testing, and Data-driven testing.

The disadvantages of Selenium WebDriver

Detailed test reports cannot be generated, RC generates detailed reports. No centralized maintenance of objects/elements. It doesn't have IDE, difficult to create test cases.

Example using Selenium WebDriver on Wikipedia

First, download "Eclipse IDE for Java Developers", Click Eclipse IDE for Java Developers, then click "INSTALL", after click "LAUNCH", it will show the eclipse neon IDE.

Second, download the Selenium Java Client Driver, then it is possible to create a new java project, also create a new package and new java class. After it needs to import the selenium libraries. Then it can write the code.

In this example is testing Wikipedia, it uses the WebDriver to get the URL, then click search input text to search "apple", in the end, close the browser, it shows in figure 2.7.

The result in console in figure 2.8.

When running the program, it shows the Firefox is opened automatically, in the input text area, insert "apple",

After some seconds, the browser jump to the "apple page",

In the end the browser is closed cause it have been written "driver.quit()" to let the browser end the operation. For now the simple test of Wikipedia by using Selenium WebDriver finished .

```java
package newpackage;

import java.io.File;


public class Myclass {
     public static void main(String[] args) {// Create a new instance of the Fir
            // Notice that the remainder of the code relies on the interface,
            // not the implementation.
         File gecko = new File("C:/Users/hp 15-ac624nl/Downloads/geckodriver-v0.
         System.setProperty("webdriver.gecko.driver", gecko.getAbsolutePath());

            WebDriver driver = new FirefoxDriver();

            // And now use this to visit Google
            driver.get("https://en.wikipedia.org/wiki/Main_Page");
            // Alternatively the same thing can be done like this
            // driver.navigate().to("http://www.google.com");

            // Find the text input element by its name
            WebElement element = driver.findElement(By.name("search"));

            // Enter something to search for
            element.sendKeys("apple");
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            // Now submit the form. WebDriver will find the form for us from the
            element.submit();

            // Check the title of the page
            System.out.println("Page title is: " + driver.getTitle());
            try {
            // Now submit the form. WebDriver will find the form for us from the
            element.submit();

            // Check the title of the page
            System.out.println("Page title is: " + driver.getTitle());
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            // Google's search is rendered dynamically with JavaScript.
            // Wait for the page to load, timeout after 10 seconds
            (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>(
                @Override
                public Boolean apply(WebDriver d) {
                    return d.getTitle().toLowerCase().startsWith("apple");
                }
            });

            // Should see: "apple - Google Search"
            System.out.println("Page title is: " + driver.getTitle());
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            //Close the browser
            driver.quit();
            }


}
```

Figure 2.7: Source code of Selenium WebDriver.

2.4.4 Capture-Replay(C&R)

“Capture-replay(C&R) web testing is based on the assumption that the testing activity conducted on a web application can be better automated by

```
1554373783852   addons.webextension.screenshots@mozilla.org     WARN    Loading e
1554373786760   Marionette      INFO    Listening on port 49421
1554373786902   Marionette      WARN    TLS certificate errors will be ignored fo
Apr 04, 2019 12:29:46 PM org.openqa.selenium.remote.ProtocolHandshake createSessi
INFO: Detected dialect: W3C
Page title is: Wikipedia, the free encyclopedia
Page title is: Apple - Wikipedia
1554373807355   Marionette      INFO    Stopped listening on port 49421
[Parent 14144, Gecko_IOThread] WARNING: pipe error: 109: file z:/build/build/src/
[Parent 14144, Gecko_IOThread] WARNING: pipe error: 109: file z:/build/build/src/
[Child 10596, Chrome_ChildThread] WARNING: pipe error: 109: file z:/build/build/s
[Child 4288, Chrome_ChildThread] WARNING: pipe error: 109: file z:/build/build/sr
[GPU 11656, Chrome_ChildThread] WARNING: pipe error: 109: file z:/build/build/src
###!!! [Child][RunMessage] Error: Channel closing: too late to send/recv, message

on/ipc_channel_win.cc, line 332

###!!! [Child][MessageChannel::SendAndWait] Error: Channel error: cannot send/rec
```

Figure 2.8: Result in console for Wikipedia by using Selenium WebDriver .

recording the actions performed by the tester on the web application GUI and by generating a script that provides such actions for automated, unattended re-execution."[2] It means it will produce the test cases simply by recording the actions on a web page, then just replay the recording to see if everything is good or not.

C&R web testing is based on capture/replay automated tools. Capture/replay tools have been developed as a mechanism for testing the correctness of interactive applications (GUI or Web applications).[27] Using a capture/replay tool, a software tester can run a Web application and record the entire session. The tool records all the user's events on the navigated Web pages, such as the key presses and mouse clicks, in a script, allowing a session to be rerun automatically without further user interaction. Finally, a test case is produced by adding one or more assertions to the recorded script. By replaying a given script on a changed version of the Web Application Under Test (WAUT), capture/replay tools support automatic regression testing.

The advantages of C&R

"C&R test cases are very easy to obtain and actually do not require any advanced testing skill. Testers just exercise the web application under test and record their actions."[2]

The disadvantages of C&R

“During software evolution the test suites developed using a C&R approach tend to be quite fragile. A minor change in the GUI might break a previously recorded test case, whose script needs to be repaired manually, unless it is re-recorded from scratch, on the new version of the web application. ”[2]

Existing tools using this technique

1. Abbot(a framework for GUI testing).

2. Jacareto(a GUI capture & replay tool supporting the creation of animated demonstrations).

3. Pounder(focused on capturing and replaying interactions for GUI testing).

4. JFCUnit(an extension that enables GUI testing based on the JUnit6 testing framework).

5. Selenium IDE (a tool for recording scripts).

.

Selenium IDE

Selenium-IDE is a tool for recording scripts, these recording scripts are Selenium test cases." the Selenium IDE (originally called the Recorder), which allows users to navigate their applications in Firefox and record their actions, forming tests. [13] Selenium-IDE is Chrome and Firefox extension, it is the most efficient way to develop test cases. It records the users' actions in the browser, using existing Selenium commands, with parameters defined by the context of that element. It helps to save time, it is a good way to learn Selenium script syntax.

Selenium-IDE is a linear script, A script that records a manually executed test instance. "Selenium IDE, Capture, and replay tests from within Firefox."[12] This script includes all the keystrokes, function keys, arrows, control keys that control the test software, and numeric keys that enter data.

Example using Selenium-IDE on Wikipedia

After installing in Firefox, it can start a recording on Wikipedia, by opening IDE, then choosing “Record a new test in a new project”, then put a name for this record, after put the Wikipedia main page URL in the BASE URL, Then start recording, just click some links and search some information, it is very easy. And all the operations are recorded by Selenium-IDE, fill in the Test Case Panel script commands, the commands are following types:

1. Click on a link - one click command

2. Enter a value - enter the command

3. Select a value from the drop-down list - select command

4. Click on checkboxes or radio buttons - click on the command

After finished the operations, it stops recording then it will show in figure 2.9 :

On the top, it will show the menu bar, where it can control how to operate the test cases. Also, it will show the Test Case Panel, it shows the recording details.

The Command, Target, and Value input fields display the currently selected command and its parameters. These are the input fields that can modify the currently selected command. The first parameter specified for the command in the Reference tab of the bottom pane is always in the Target field. If the second parameter is specified by the Reference tab, it is always in the Value field. The Comment field, which is allowed to add comments to the current command for later reading.

To save a Selenium IDE project, click the save button at the top right-hand side, Selenium IDE will download a .side file, which contains all test cases and suites.

Then it is possible to run the test cases, by clicking ” Run current test”. Here are also some special ways to run, for example, “Stop in the Middle”, in this one setting break point, which is good for debugging. Also “Start from the middle”, “ Run any Single Command”, and so on.

It will show the testing result in the Log panel, it will show the executing procedure. Until now the testing by using Selenium IDE on Wikipedia finished .

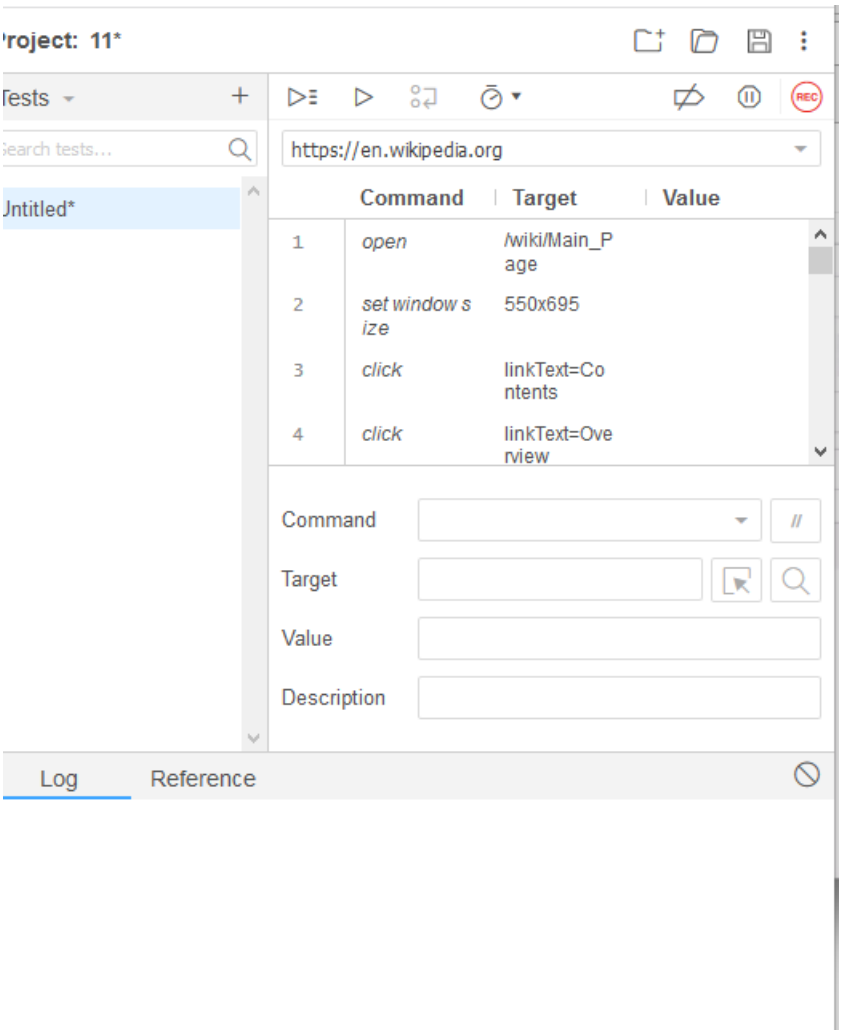


Figure 2.9: Selenium-IDE Capture-Replay .

2.4.5 Visual GUI Testing

Visual GUI Testing(VGT) is an interactive graphical software testing methodology, it verifies if the image appears correctly to users. "Visual GUI testing is a script based testing technique that uses image recognition, instead of GUI component code or coordinates, to find and interact with GUI bitmap components, e.g. images and buttons, in the SUT's GUI. "[14]

The test tool can visually identify (see) the position on the screen to interact with just like a manual tester. "Visual GUI Testing is performed to acquire data about the technique's maintenance costs and feasibility."[15] It is defined as "a tool-driven test technique where image recognition is used to

interact with, and assert, a system's behavior through its pictorial GUI as it is shown to the user in user-emulated, automated, system or acceptance tests".

"Testers can write a visual test script that uses images to specify which GUI components to interact with and what visual feedback to be observed. "[16] The programmer can use a visual GUI testing tool to examine and manipulate the system under testing.

GUI interaction based on image recognition allows visual GUI testing to mimic user behavior, treat the SUT as a black box, whilst being more robust to GUI layout change.

It is therefore a prime candidate for better system and acceptance test automation. However, the body of knowledge regarding visual GUI testing is small and contain no industrial experience reports or other studies to support the techniques industrial applicability.[14] A realistic evaluation of industrial-scale testing problems is key in understanding and refining this technique.

The body of knowledge neither contains studies that compare different visual GUI testing tools or the strengths and weaknesses of the technique in the industrial context.

For VGT the important point is image recognition allows the technique's tools to automate test cases that previously had to be conducted manually with equivalent input and output to a human user. VGT tools can emulate end-user behavior on almost any GUI-based system, regardless of implementation language, operating system, or platform.

In summary, Visual GUI Testing is a technique that gives the user the ability to transition from manual testing into automated testing, lower lead times, and create a culture of continuous software integration, development, and delivery.

Advantages of Visual Testing

Visual GUI Testing is applicable to any GUI driven AUT, due to the use of image recognition.[5] It is not expensive, and easy to understand and explain. More efficiency and flexible.

Disadvantages of Visual Testing

Suffer from false test results due to possible image recognition failure.

Existing tools using this technique

1. eggPlant is a black-box GUI test automation tool.
2. AutoIt is an automate Windows GUI and general scripting tool.
3. Ranorex is a commercial Windows GUI test automation tool.
4. Sikuli is an open source framework to automate GUI testing tool.

Sikuli

"Sikuli is a visual approach to search and automation of graphical user interfaces using screenshots."[17] It is based on image retrieval technology, it provides a set of Jython-based scripting language and integrated development environment. "Sikuli allows users to take a screenshot of a GUI element (such as a toolbar button, icon, or dialog box) and query a help system using the screenshot instead of the element's name." [17] Sikuli also provides a visual scripting API for automating GUI interactions, using screenshot patterns to direct mouse and keyboard events.

Sikuli Script, a scripting system that enables programmers to use screenshots of GUI elements to control them programmatically. [17] The system incorporates a full-featured scripting language (Python) and an editor interface specifically designed for writing screenshot based automation scripts.

Sikuli allows users or programmers to make direct visual reference to GUI elements.[17] To search a documentation database about a GUI element, a user can draw a rectangle around it and take a screenshot as a query.

The real-time image retrieval of the current screen captures the operational object, simulates the user's behavior, and matches the screen area to verify the true visual display. The user can use the screenshot to directly reference the GUI elements to the program and complete the interaction.

Example using Sikuli on Wikipedia

After installing Sikuli , it will show IDE in figure 2.10:

In the top, there is menu bar, it is possible to do some operations like :

- "Take screenshot"
- "Insert image"
- "Region"

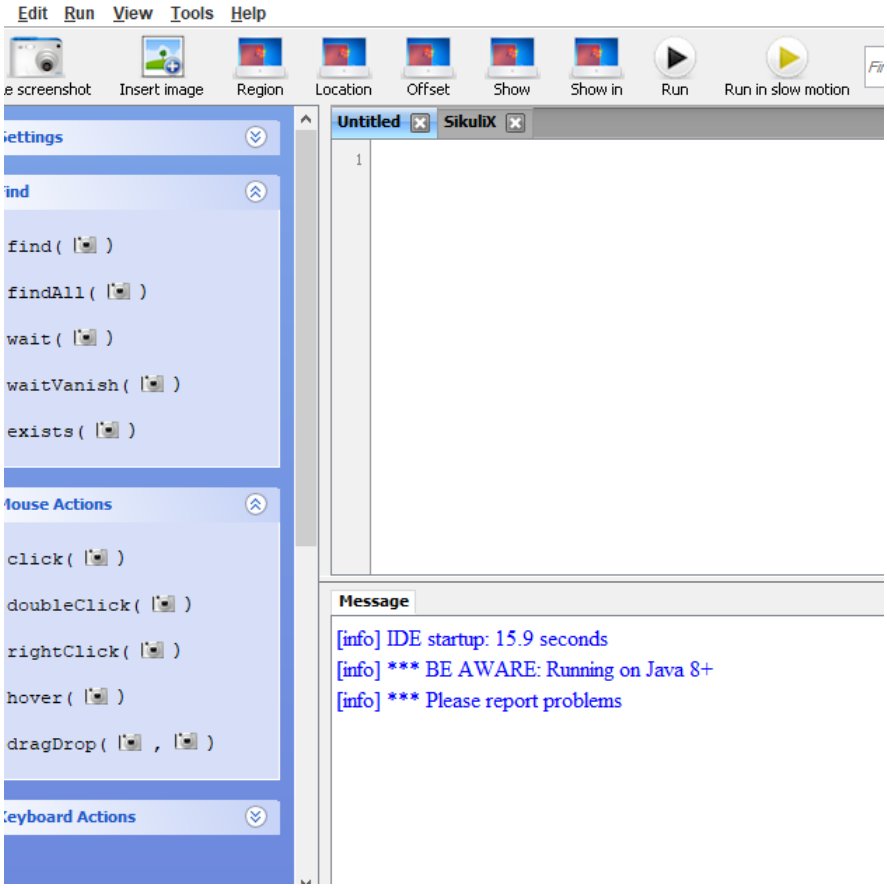


Figure 2.10: Example using Sikuli on Wikipedia.

- “Run”
- “Run in slow motion”

In the left part , there are commands to operate on the web page, like :

- “find()”
- “findAll()”
- “wait()”
- “click()”
- “doubleclick()”
- “type(text)”

In the lower part It will show “Message”,when running the test , the execution result will come out.

Simple example in figure 2.11,

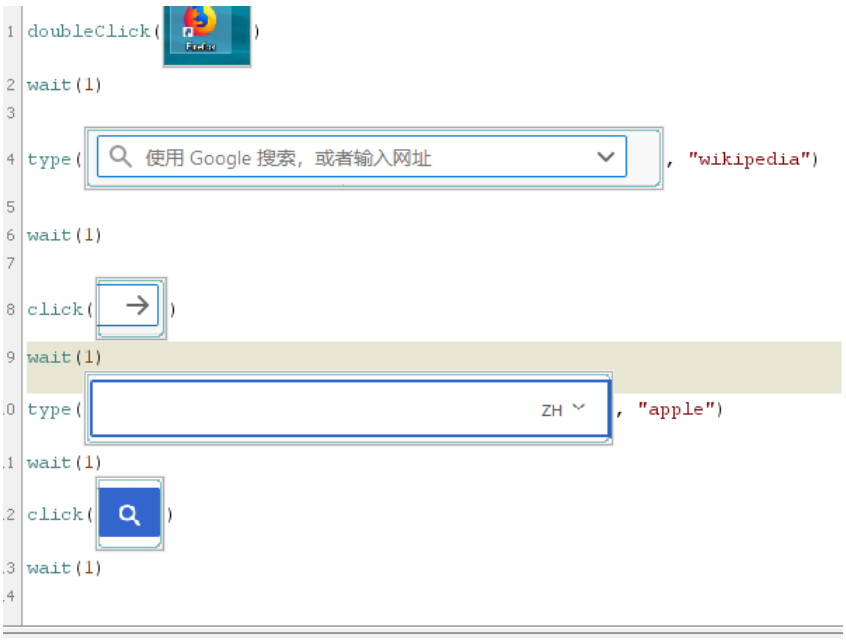


Figure 2.11: Example code using Sikuli on Wikipedia.

It double clicks Firefox, on the search bar it types “Wikipedia”, then clicks “enter”, it will jump to Wikipedia page, after it searches “apple”, then clicks “enter”, it will jump to the page about “apple” which contains many different meanings of “apple”.

In the “Message” , it will show the “log information” in figure 2.12,

For now the example for Wikipedia finished.

2.4.6 Exploratory comparison of testing techniques

Model-based testing automatically creates the test model, reducing the escaped defects compared with other testing approaches. This method is good for the project which is not so complicated, cause for a complicated project, it is very difficult to create the model.

Scripted based Testing is very powerful cause it uses scripting language syntax which is simple and flexible, but it lacks the strict compilation process, variable definitions, function definitions, etc. So it is more likely to occur problems which will trouble the testers. In general, this technique is very

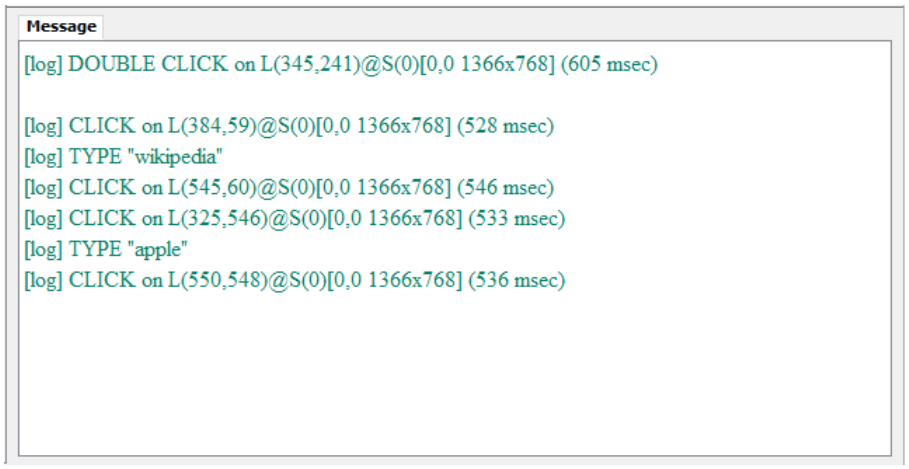


Figure 2.12: Example log information using Sikuli on Wikipedia.

efficient for the processing of multiple files, data flow and processing are more complicated.

Capture-replay(C&R) web testing is very easy to get and do not require any advanced testing skill, just record the actions. But it is very weak for testing, any little change in GUI will influence the previous recording test cases, it just useful for a very simple project.

Visual GUI Testing is applicable to any GUI driven AUT, due to the use of image recognition. It is not expensive, and easy to understand and explain. More efficient and flexible. Suffer from false test results due to possible image recognition failure.

2.5 Maintenance and fragility

2.5.1 Fragility

Testing fragility is defined as when it fails or needs maintenance due to the natural evolution of the AUT, which is a very important issue for web application testing. A little failed test case will show a big problem for the system design, needs a deep investigation to find out what causes this failure. "We define the concept of fragility of test classes, and provide metrics to estimate the fragility of a project by automated inspection of its test suite."[7] A set of possible reasons that will cause frailties specific to web testing will be introduced.

"A GUI test class is said to be fragile when:

- it needs modifications when the application evolves;
- the need is not due to the modification of the functionalities of the application, but to changes in the interface arrangement and/or definition."[7]

When it fails or requires maintenance due to the natural evolution of the application under testing, provided the specific capabilities it produces have not been altered.

The heuristic can be seen as a second-order stress test to detect nonlinearities in the tails that can lead to fragility.[23] Fragility testing is that it should not affect the change of test results under any circumstances. "Any time a pre-existent method of a GUI test class is modified it is assumed the change is due to test fragility." [7] The example most encounter is when changing a part of the code and break a test that should not be broken. However, there may be other influencing factors: test data, current date and time, or part of other contexts or other tests. As long as these factors affect the test results, there will be Fragility tests.

2.5.2 Maintenance

Website maintenance is the regular checking of websites for issues and errors and security updates to keep them relevant. To keep the website healthy, drive continued traffic growth, it needs to do this consistently.

"Keeping a website well maintained is important for big and small businesses to attract and retain their customers."[8] For companies, Website maintenance tends to be one of these issues because it doesn't always cause immediate problems. The invention relates to the maintenance of a website. In particular, the invention relates to gathering website user help data and modifying the website through the use of a feedback loop using the gathered data.[24] However, just as it can be too long without regular checking, it can damage the website.

The things should be done for maintenance:

- Renewing the domain name of website,make sure that all domain names are renewed in a timely manner.
- Test the loading speed of website. If it takes a long time to load, it needs to improve.
- Update the core plugins and website software,Update dates and copyright notices.

- Check the website for 404 errors and if there are any, fix or redirect them.
- Security updates and bug fixes. Make sure web developer and hosting provider update the software and install upgrades, security patches, bug fixes or any other updates that may compromise the operating system, web server, database.
- Analyze the security scans and if there are any problems, make sure they're resolved. Set aside time to methodically and thoroughly review all pages of the website.

Categorize the changes in a test code under four categories:

- Perfect maintenance, when a test code is upgraded to increase quality (e.g., to increase coverage age or adopt well-known test patterns);
- Adaptive maintenance, to make the test code evolve in line with the evolution of the production code;
- Protective maintenance, to alter parts of the code that may require intervention in future notifications;
- Corrective maintenance, to perform bug fixing. According to the definition of GUI testing sensitivity, adaptive maintenance is important.

Website Maintenance is very important. "With regular website updates and maintenance, the business will attract and retain customers, offer useful content, and maintain good search engine rankings."[8]

For this thesis, some maintenance metrics for web application testing are defined.

Chapter 3

Empirical experiment with dolibarr

In this chapter, the experiment with web application dolibarr will be done, the purpose is writing some test cases to test the application, and defining some metrics to analyze the maintenance effort during the application evolution. To make some good suggestions for web application testing.

3.1 Context

The work is the following steps:

1. Writing the test suites for the web application dolibarr;
2. Count the modified lines for the evolution of the application;
3. Make a characterization of the fragility issue and an estimation of its occurrence in a open-source project.

Defining some metrics for the evolution, "How much are GUI test classes associated with the analyzed set of tools modified through consecutive releases of an open-source Android project?" [6] also for the fragility, "How fragile are GUI test classes associated with the analyzed set of tools to modifications performed on open-source Android projects and their graphical appearance?" [6]

Five released versions source code of the application are cloned from GitHub, then 15 test cases for the earliest release application are written, then it should update the application, also need to modify the test cases

for adopting the application. It compares file-by-file between consecutive releases. And the modifications of each test case are recorded. Then using "Test evolution" metrics and "Fragility of test cases and methods" metrics.

3.2 Introduction Dolibarr

Dolibarr is an ERP / CRM system, it is a modern software package to manage your company or foundation activity (contacts, suppliers, invoices, orders, stocks, agenda, accounting, ...).

It's open-source software (wrote with PHP language) designed for small and medium companies, foundation, and freelances. It is easy to use, study, modify or distribute according to its Free Software license.It is possible to use it as a standalone application or as a web application to be able to access it from the Internet or a LAN.

On Github[1], there are three "Setup" ways to install Dolibarr,

- Simple setup: packaged version
- Advanced setup: standard version
- Saas/Cloud setup: commercial version

according to different purpose to install it.

Main application/modules (all optional):

- product and service catalog
- inventory management
- bank account management
- customer directory
- order management
- business proposal
- contract management
- invoice management

[1]https://github.com/Dolibarr/dolibarr

- invoice and payment management
- manufacturing expense list
- Transportation
- plug and play

Dolibarr is written in PHP and supports all versions higher than 5.5.0+ As it shows below, there are many functions on the dolibarr application,

- Customer prospect or supplier directory features
- Product and service management capabilities
- Inventory management functions
- Bank account management functions
- Business Operations Management Functions
- Order management functions
- Contract management functions
- Invoice management functions
- invoice and payment management
- Payment management functions
- Inquiry management function
- Logistics management functions
- Expense report management function
- Following social and fiscal tax payments feature
- EDM (electronic document management) function
- Employee leave function
- Mass mailing function
- Realize surveys

- POS function
- Donations management
- Report function
- The ability to generate PDF documents (invoices, proposals, orders, etc. to generate PDF documents ...)
- Import and export tools (CSV or Excel)
- Bookmark management function
- LDAP connection capabilities)

It is possible to click them to do some opeartions.
It shows "Home" page of Dolibarr in figure 3.1,

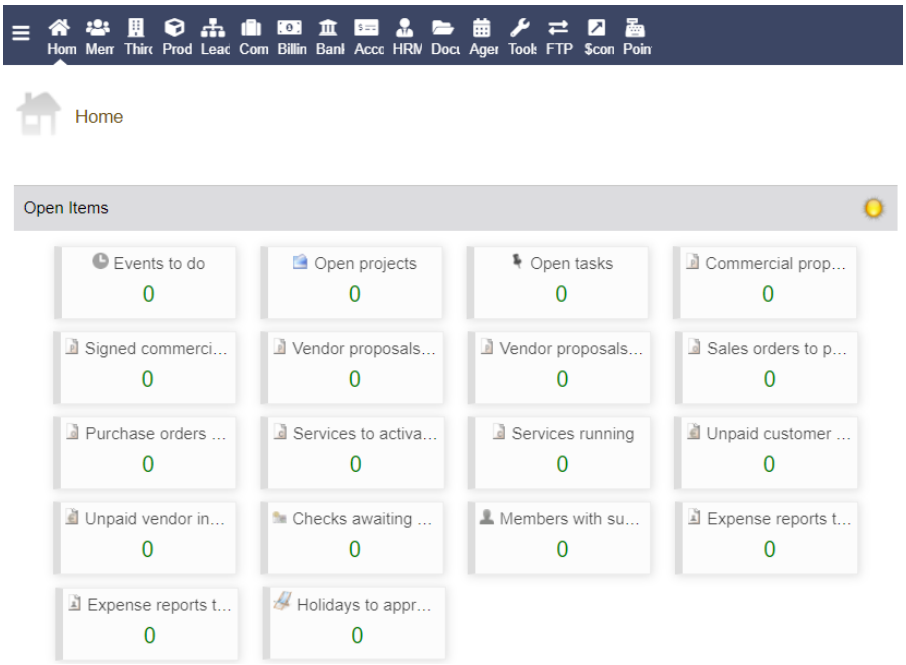
3.3 Test cases

15 test cases for 5 versions of dolibarr are developed. According to web application, some test cases for most parts of web functions are defined, like create new customer,supplier or products, add, delete or modify the information and so on. All the test cases show in the table 3.1:

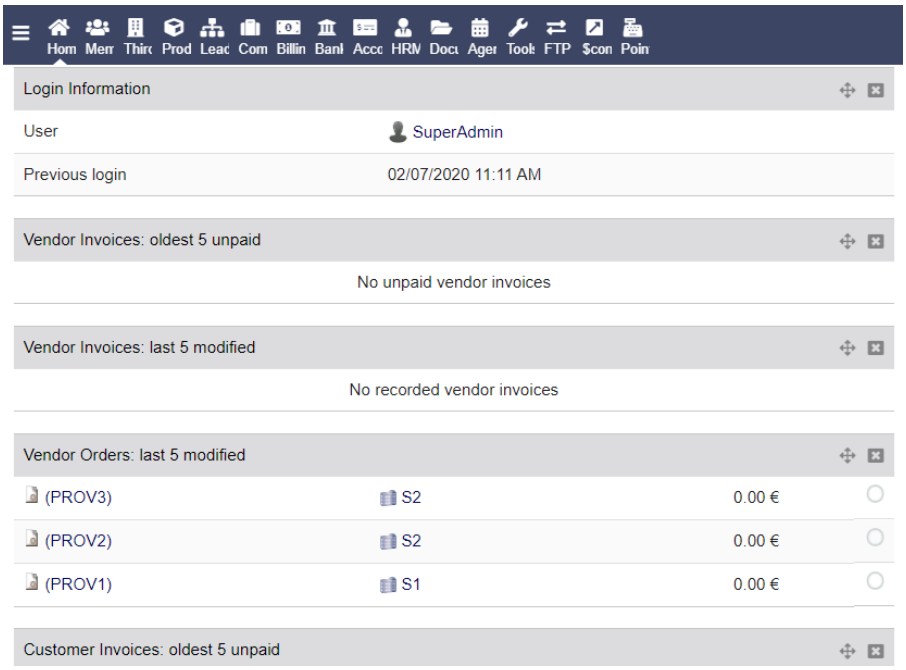
3.4 Use case template for TC1 (new customer)

All test cases were defined and documented by using use case templates."A use case model defines the functional scope of the system to be developed. Attributes of a use case model may therefore serve as measures of the size and complexity of the functionality of a system."[25] An use case is a description of how the system responds to external requests in software engineering or system engineering, is a technology to obtain requirements through user use scenarios."Use cases are assumed to be developed from scratch, be sufficiently detailed and typically have less than 10–12 transactions."[25] Each use case provides one or more scenarios that illustrate how the system interacts with the end user or other systems, that is, who can use the system to do something to achieve a clear business goal.

Avoid using technical terms when writing use cases. Instead, use the language of the end-user or domain expert.[30] Use cases are typically authored



(a) Dolibarr page 1.



(b) Dolibarr page 2.

Figure 3.1: Dolibarr page.

Table 3.1: Test cases table.

Test Cases	Description
TC1	Login Test: Test if login successfully
TC2	New Customer: Create a new customer for company
TC3	New Supplier: Create a new supplier for company
TC4	Modify Customer:Modify some information for one customer
TC5	Add Bank for Customer: Add bank information for customer
TC6	Delete Customer: Delete one customer
TC7	Check all links: Click all links on to do page
TC8	Change setup: change some setup for display
TC9	New Product: Create a new product
TC10	New Project: Create a new project
TC11	New Leave: Create a new leave
TC12	New Service: Create a new service
TC13	New Task: Create a new task
TC14	Modify Product: Modify information for one product
TC15	New Invoice: Create a new invoice

by software developers and end-users. There are many ways to write a use case in the text, from use case brief, casual, outline, to fully dressed, etc. And with varied templates. Writing use cases in templates devised by various vendors or experts is a common industry practice to get high-quality functional system requirements.

Write detailed or complete use cases. There is no universal template. There are many competing templates. At the same time, programmers are encouraged to use templates that are suitable for their work or their projects.

Compared to the details of a specific template, the standardization of the project is much more important, but the key parts of these templates are Roughly the same, so although there are differences in some terminology or other aspects, these use cases are essentially the same.

Typical sections include:

- Use case name
- Roles
- description

- precondition
- Event flow
- Elementary stream
- Alternative stream
- Post condition
- Extension point
- Business Rules
- special requirement
- Iteration

Different templates often have other parts, such as:

- assumptions
- abnormal flows
- recommendations
- industry details

In this section it is possible to create a use case template for a single test case,that is "New Customer" in table 3.2.

3.5 Metric Definition

To count the modifications in the test code, It is possible to define some metrics, with the goal of describing change histories in the source files. “Tdiff” is used to count the number of line codes added, deleted, modified. “Pdiff”, the amount of added,deleted, or modified lines of code on which are based on several of the metrics, can be computed as the sum of LA (Line of code added) and LD(line of code deleted) for all the files of the release.

Here is the list of definitions of the metrics about “Test Evolution”(“describe the evolution of open-source projects and the respective test suites; they have been computed for each release or for each couple of consecutive tagged releases.”[6]):

- TTL: Total tool LOC(line of code), is the number of lines of code belonging to classes.
- NTC: Number of tool classes.
- TLR: Tool LOCs ratio is defined as:

$$TLRi = \frac{TTLi}{Plocsi};$$

Where $Plocsi$ is the total number of application LOCs for release i. This metric, the range in the [0, 1] interval, allows to quantify the relevance of the testing code associated with a specific tool.

- MTLR: Modified tool LOCs ratio is defined as:

$$MTLRi = \frac{Tdiffi}{Tlocsi - 1};$$

Where $Tdiffi$ is the number of lines added, deleted,modified in the application from release $i - 1$ to i; "This quantifies the amount of changes performed on existing LOCs that can be associated with a given tool for a specific release of a project."[6]; If this value is more than 1, which means a large amount lines added, deleted and modified in the application evolution.

- $MRLTi$: is defined as:

$$MRTLi = \frac{Tdiffi}{Pdiffi;}$$

$Tdiffi$ and $Pdiffi$ are, respectively, the number of lines added, deleted, or modified tool and production LOCs in the transition between releases $i - 1$ to i. It is computed only for releases featuring code associated with a given testing tool (i.e., $TRLi > 0$). This metric lies in the [0, 1] range, and values close to 1 imply that a significant portion of the total code churn during the evolution of the application is needed to keep the test cases written with a specific tool up to date.

- $TMRi$: Tool modification relevance ratio is defined as:

$$TMRi = \frac{MRTLi}{TLRi - 1};$$

"This ratio can be used as an indicator of the portion of code churn needed to adapt classes relative to a given testing tool during the evolution of the application."[6] Only computing this value when $TLRi - 1 > 0$. When this value is greater than 1, this metric is an indicator that the amount of work required to change the test code is greater than the actual relevance of the test code to the test code. Also considering the lower value of this indicator makes it easier to prove code associated with a specific test device for changes device under test.

- TCV: Tool class volatility can be computed as:

$$TCVj = \frac{Modsj}{Lifespanj};$$

"where $Modsj$ is the amount of releases in which the class j is modified, and $Lifespanj$ is the number of releases of the application featuring the class j."[6].

For the "Fragility of Test Classes and Methods", from the metrics defined in the previous part, the information about the modified methods and classes can be obtained, based on these data, it can define the metrics about the approximated characterization of the fragility of test suites.

Modified classes can be three different ways : first, modify some unimportant things like comments,imports,declarations; second, add some test methods; Last, Delete some test methods. Also combinations of these three ways.

"Additions and removals of test methods are considered the consequence of a new functionality or a new use case of the application; hence, they are not considered as evidence of fragility of test classes."[6] In the meanwhile, when modifying the test method, it indicates the changes of the application, so make the test classes that contain them as fragile according to the definition.

So the definition of Modified tool classes ratio(MCR),

$$MCRi = \frac{MCi}{NTCi - 1};$$

"where MCi is the number of classes associated with a given testing tool that are modified in the transition between releases $i - 1$ and i, and $NTCi - 1$ is the number of classes associated with the tool in release $i - 1$(the metric is not defined when $NTCi - 1 = 0$). The metric lies in the [0, 1] range: the larger the values of MCR, the less the classes are stable during the evolution of the app"[6])

the table of the definition in table 3.3.

3.6 Procedure

In this section, it will introduce how to calculate the metrics defined before. "In the exploration of the history of Android repositories, the versions that have been considered for tracking the evolution of test classes are the tagged points of release histories."[6] To use git diff to get the differences between two consecutive versions of the application, using "cloc" command to the total lines of code of the application or test cases. After getting these values, it can calculate TLR, MTLR, MRLT, and TMR for each test tagged release of any project.

For the work, it cloned five release version of dolibarr, they are version 6, version 7, version 8, version 9, version 10. It defines 15 test suites for version 6, after starting to update the version 6 to version 7, then it needs to modify the test suites to make the testing work properly.

The data for the application and the test suites in table 3.4.

The data modified during the evolution of release version in table 3.5.:

3.7 Results and Discussion

In this part, it will give the result which gets from the previous procedure.

A program are written which will calculate the results , it is a bash script. The content of code is: it reads the pathnames of the test cases and application(the input are pathnames), using 'Cloc'[2] to calculate the number of lines respectively for different versions(contains different versions for test cases and application), it stores these values and calculates 'number lines modified in test cases' during evolution, also calculates 'number lines modified in application' during evolution.

For TLR, MTLR, MRTL, the program contains the code part according to the definition of these metrics, also for TMR. There is a code part also for the average values of TLR, MTLR, MRTL, and TMR. In the end, it will output the values and average values of TLR, MTLR, MRTL, and TMR.

A pseudo code shows in figure 3.2.:

The results of the metrics in table 3.6.

From the table, some conclusions can be obtained,

- For TLR, it shows that the relevance of the testing code associates with

[2]http://cloc.sourceforge.net/

the application, as the application is very huge, while the test cases are very small, so only testing very little parts of the application.

The average value of TLR is 0.084%, which is a very small value, this should be considered as a consequence of small test suites.

Also, it decreases from version 6 to version 7, this means for the evolution, the application has added more functions and more modules becomes more strong and powerful.

- For MTLR, it shows the number of changes performed on the existing LOCs that can be associated with a given tool for a specific release of a project, if this value is larger than 1, this means more lines added, deleted and modified in test classes in the transition between two consecutive releases, In the results, the average value of MTLR is 8%, so around 8% of test cases code have been changed between two consecutive releases.

  From the result, it shows from version 6 to 7, the value is 7%, is very small, so this means small modified amount LOCs will lead to a small value of MTLR value. In the meanwhile from version 7 to 8, the value of MTLR is 16%, so from this evolution, the application changed a lot, so the test cases must also change according to the application.

- For MRTL, this value is in the [0, 1] range, and values close to 1 imply that a significant portion of the total code churn during the evolution of the application is needed to keep the test cases written with a specific tool up to date.

  The average value of MRTL is 18.5%, this means in every two consecutive releases, around 20% of code in the application has been modified.

  From version 6 to 7, this value is 40%, which is very big, this means in this updating a big amount number of code in the application changed.

  While from version to version 9, MRTL is only 7%, it shows only slight modification in this evolution.

- For TMR, this is computed only when $TLRi - 1 > 0$. It considers a value greater than 1 of this metric as an index of greater effort needed in modifying the test code than the actual relevance of testing code with respect to the modification of application code.

  It speculates lower values of this indicator as evidence of easier flexibility of code associated with a specific test device for changes in the AUT.

In general , “those values imply that the amount of churn needed for the code associated with a specific testing framework is not linear with the relative amount (with respect to total production LOCs) of such code inside the application”[6].

The result table of TCV in table 3.7.

- TCV: The average value for TCV is 0.42. which signifies the wonder of variability from the point of view of a view of the individual classes associated with a particular test framework, clarifying that each test class must modulate, on average, every ten tagged notifications in which it appears.

  The value is small means this test case is very stable. While if the value is large means every time it needs to modify this test case, it is not powerful for the work.

Also plotting the trend for TLR in figure 3.3:

From the plot, it shows that the trend of TLR is decreasing during the version evolution, it means the relevance of the test cases code associated with dolibarr is going down, that's because during the application evolution, the developer add new functions and new parts for the application, but the test cases was start from the earliest version, so it only according to the old version, write the test cases, it can't cover so much parts for the application, so as the updating of the application, it appears more new things. so TLR is decreasing during the evolution.

Also plotting the trend for MTLR and MRTL in figure 3.4:

From the plot, it can compare the trend of MTLR and MRTL, it shows that from version 6 to version 7, there is a big jump for MRTL, which means there was a significant portion of the total code changed during the evolution of the application is needed to keep the test cases written with dolibarr up to date.

While MTLR doesn't change too much, this is because the changes in the application are mainly about adding new functions, while in the test cases doesn't contain these new functions, so test cases don't change too much, so MTLR is just slightly increasing a bit.

Then from version 7 to version 8, MRTL decreases, this means the developer is just maintaining the application, so the MRTL is slightly going down.

And also from version8 to version 9, from version 9 to version 10, this rate doesn't change too much, this means the application is very stable during the evolution, only doing some basic maintenance for the application.

While for MTLR, its value is stable, doesn't change a lot during the application evolution, it is not a good phenomenon, if the application changed a lot while the test cases don't change according to the application, which means out test cases is not very related to the application, it should pay attention about this, and write the test more relative to the application.

Also plotting the number of modified code of test cases in version evolution in figure 3.5:

From the plot, the red color is the number of code for test cases, while the blue color is the number of modified code of test cases in version evolution.

It can be observed that during every evolution, the code modified in the test cases code is occupied around 8 %. in version 7 to 8, its value is higher.

During the other code release, the code modified is less. It can be noticed that during every evolution the amount of code changed in the test case is not a big number, this is a normal situation, cause the application changed, while for the test cases it only modifies them to keep be consistent with the application. For the developer, it is important to make the test cases fit for the application.

Also plotting the number of modified code of application in version evolution in figure 3.6:

From the plot, the red color is the number of code for the application, while the blue color is the number of modified code of the application in version evolution.

From the result, it shows that in the evolution version 6 to version 7, the amount changed in the application is a big number, occupied around half of the amount of the application, this shows that in this evolution application changed a lot(Add the new functions and modified some parts).

While in the evolution version 7 to version 8, the code modified is around 13%, it is slightly modified during the evolution. While in version 8 to version 9, the number of code modified is small, is only 7%. And for the version9 to version 10, it is around 14%.

Also it can be noticed that the number of application code is increasing during the evolution, which means the application is more powerful and stronger. It a good thing for the user to have a nice experience when using the web application.

Also plotting the trend for TMR in figure 3.7:

From the polt, it shows the average of this value is 201.5, From version 6

to 7, this value is big, is 400, this is because in this version evolution, TLR is small. While from 8 to 9, this value is very small, is only 87, in this evolution, TLR is higher.

Table 3.2: Use Case table.

Test Cases	**Description**
Use case	Create a new Customer
Scope	Third parties
Level	User Goal
Intention in context	Add a new customer information in the database
Primary actor	–
Support actor	–
StateHolders' interest	–
Precondition	Already known the information of customer
Minimum guarantees	–
Success guarantees	–
Trigger	–
Main Success scenario	1. Click the web page, and login. 2. Click the "Third parties",then click "New Customer". 3. Insert information, "Thirds-party name","Alias name",choose "Customer" from "Prospect/Customer" box, select "No" from "Vendor" box, insert "Bar code", "Address", "Zip code","City","Country","State/Province","Email", "Web","Phone","fax",these information. 4. Then click "CREATE THIRD PARTY". 5. Click "List of Customer",find the customer.
Extensions	• 4a. Web failure of "insert customer wrongly". • 4a1.System reports failure to user with advice, back up to previous step. • 4a2. User either back out of this use case, or tries again. • 5a. Website does not return the needed information. • 5a1. Reload the webpage to update.

Table 3.3: Metric Definition.

Group	Name	Explanation	Type	Range
Size	NTC	Number of Tool Classes	Integer	[1,∞)
	TTL	Total Tool LOCs	Integer	[1,∞)
Test Evolution	TLR	Tool LOCs Ratio	Real	(0,1]
	MTLR	Modified Tool LOCs Ratio	Real	[0,∞)
	MRTL	Modified Relative Tool LOCs	Real	[0,1]
	TMR	Tool Modification Relevance Ratio	Real	[0,∞)
	TCV	Tool Class Volatility	Real	[0,∞)
Fragility	MRC	Modified Tool Classes Ratio	Real	[0,1]

Table 3.4: Data for Application And Test Suites

lines of code|version	Version 6	Version 7	Version 8	Version 9	Version 10
Test Cases	928	941	943	946	957
Application	698221	999572	1113041	1133130	1300839

Table 3.5: Data Modified for Application And Test Suites

Modified lines|version	Version 6-7	Version 7-8	Version 8-9	Version 9-10
Test Cases	69	152	106	102
Application	321351	133469	23089	167709

Table 3.6: Results Metrics

Version|Metrics	TLR	MTLR	MRTL	TMR
Version 6	0.1%	–	–	–
Version 7	0.09%	7%	40%	4
Version 8	0.08%	16%	13%	1.44
Version 9	0.08%	11%	7%	0.87
Version 10	0.07%	10%	14%	1.75

```
Initialize pathname_of_testcases=$1
Initialize pathname_of_applications=$2

LOOP

Go to the testcases directory
        for different versions
        Use "Cloc" count the lines of testcases
        and store it for "Previous_version_linescodefortestcases_i"

        Go the next version
        Use "Cloc" count the lines of testcases
        and store it for "Current_version_linescodefortestcases_i+1"

        calculate the difference between two continues version:
        Set difference_fortestcases_i_i+1 = Current_version_linescodefortestcases_i+1 -
Previous_version_linescodefortestcases_i



Go to the applications directory
        for different versions
        Use "Cloc" count the lines of applications
        and store it for "Previous_version_linescodeforapplications_i"

        Go the next version
        Use "Cloc" count the lines of applications
        and store it for "Current_version_linescodeforapplications_i+1"

        calculate the difference between two continues version:
        Set difference_forapplications_i_i+1 = Current_version_linescodefortestcases_i+1
- Previous_version_linescodefortestcases_i




Set TLRi = Previous_version_linescodefortestcases_i /
Previous_version_linescodeforapplications_i
Set SUM_TLR =   SUM_TLR + TLRi
Set NUM_TLR++

Set MTLRi_i+1 = difference_fortestcases_i_i+1 / Previous_version_linescodefortestcases_i
Set SUM_MTLRi_i+1 = SUM_MTLRi_i+1 + MTLRi_i+1
Set NUM_MTLR++

Sst MRTLi_i+1 = difference_fortestcases_i_i+1  / difference_forapplications_i_i+1
Set SUM_MRTLi_i+1 = SUM_MRTLi_i+1 + MRTLi_i+1
Set NUM_MRTL++

Set TMRi_i+1 = MRTLi_i+1 / TLRi
Set SUM_TMRi_i+1 = SUM_TMRi_i+1 + TMRi_i+1
Set NUM_TMR++

Set AVG_TLR = SUM_TLR / NUM_TLR

Set AVG_MTLR = SUM_MTLRi_i+1 / NUM_MTLR

Set AVG_MRLT = SUM_MRTLi_i+1 / NUM_MRLT

Set AVG_TMR = SUM_TMRi_i+1 / NUM_TMR++
```


Figure 3.2: Pseudo code for calculating Metrics.

Table 3.7: Results TCV

Test Cases	**Modified**	**LifeSpan**	**TCV**
TC1	0	5	0
TC2	4	5	0.8
TC3	3	5	0.6
TC4	2	5	0.4
TC5	3	5	0.6
TC6	0	5	0
TC7	1	5	0.2
TC8	2	5	0.4
TC9	3	5	0.6
TC10	4	5	0.8
TC11	0	5	0
TC12	4	5	0.8
TC13	3	5	0.6
TC14	0	5	0
TC15	2	5	0.4

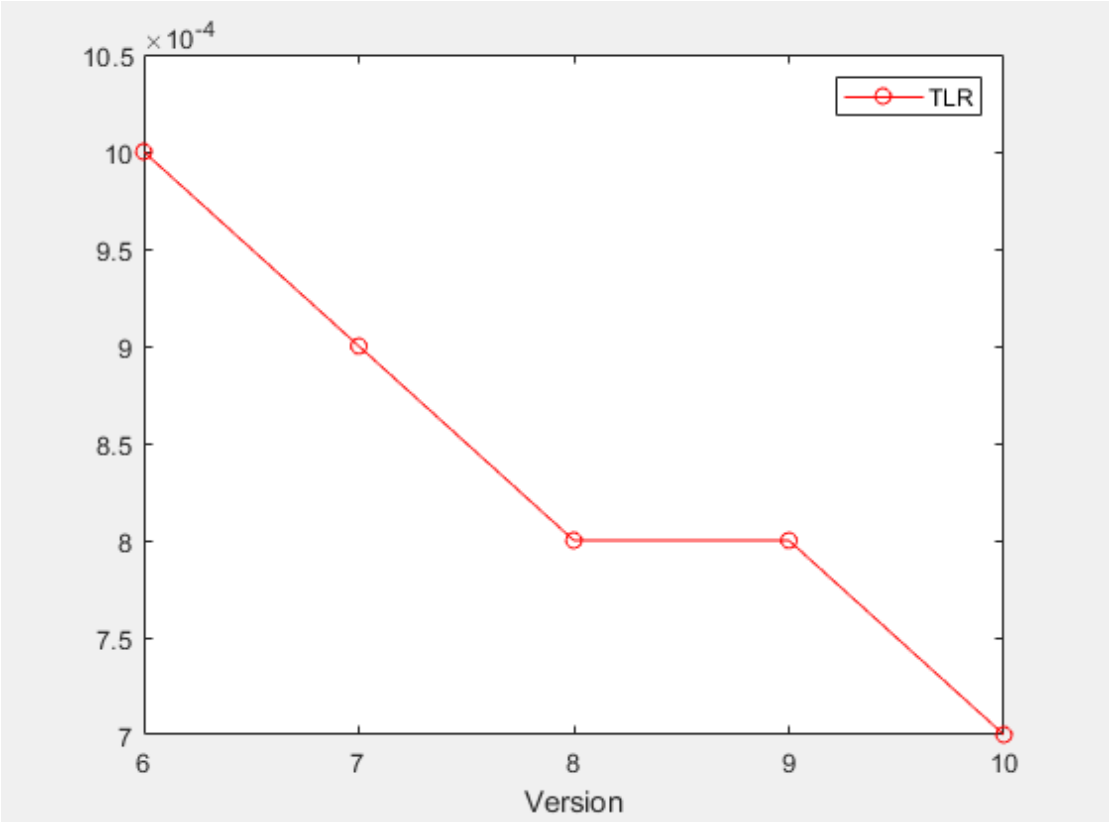


Figure 3.3: The trend of TLR.

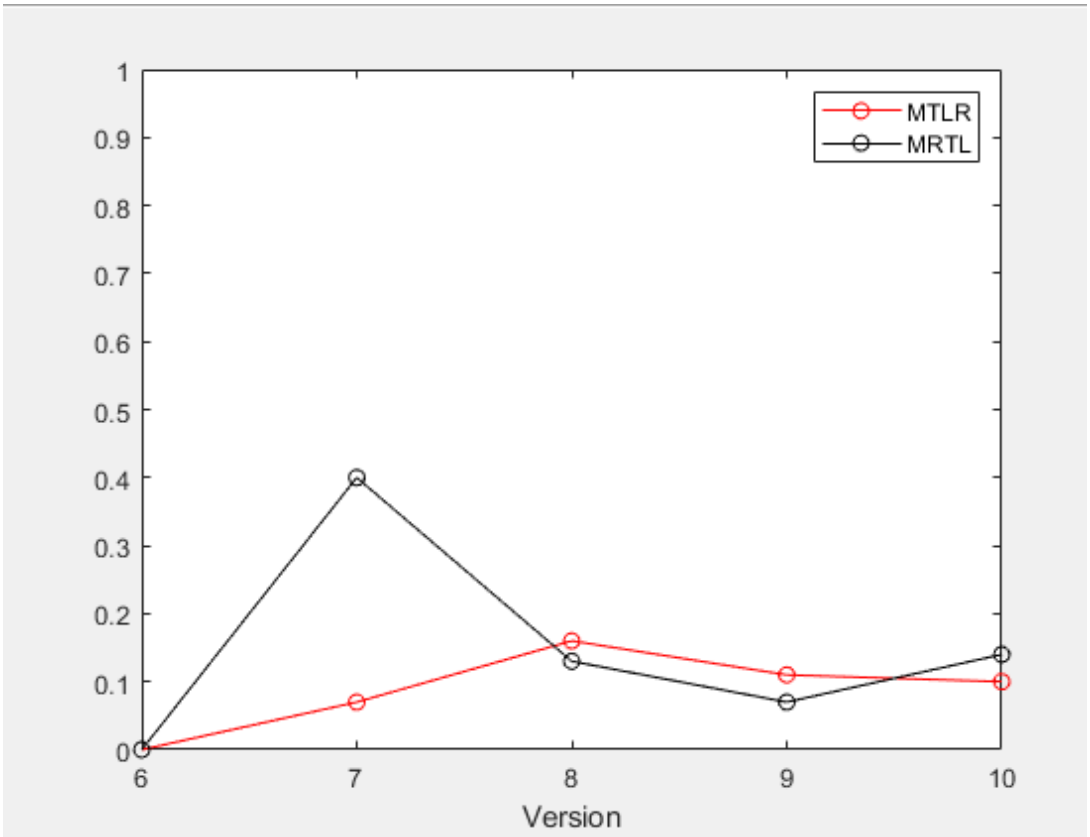


Figure 3.4: The trend of MTLR and MRTL.

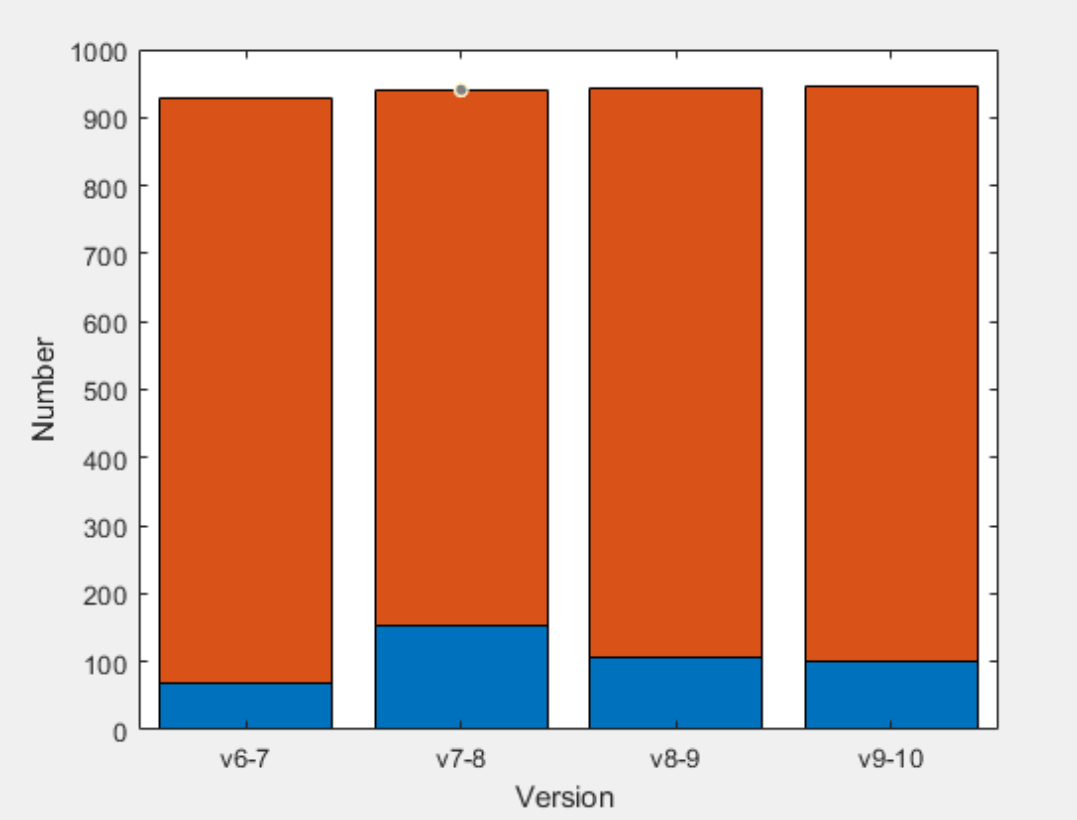


Figure 3.5: The number of modified code of test cases in version evolution .

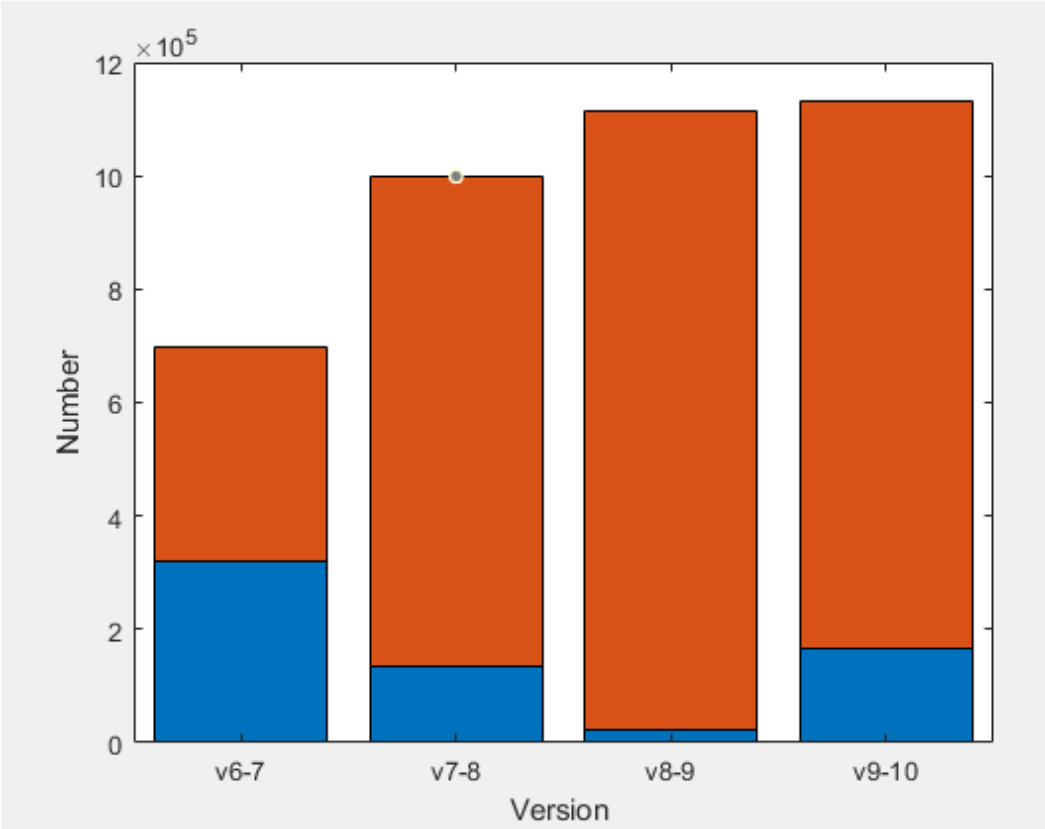


Figure 3.6: The number of modified code of application in version evolution.

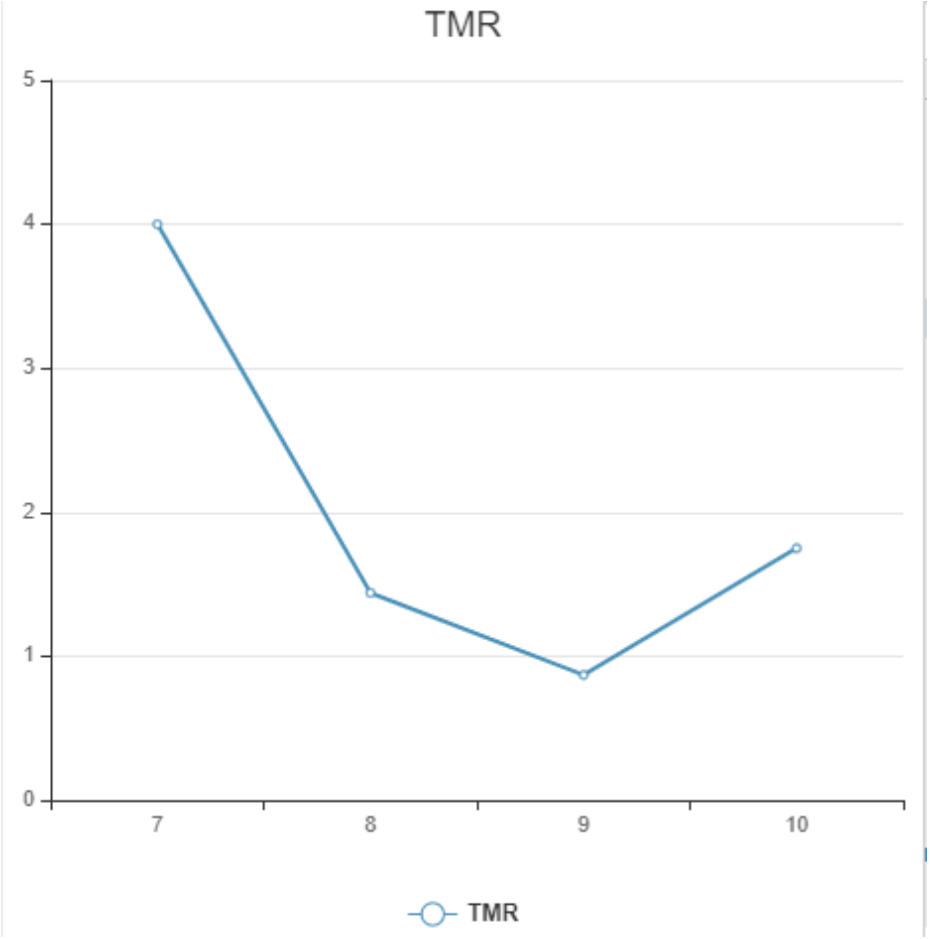


Figure 3.7: The trend of TMR.

Chapter 4

Conclusion and Future Work

This chapter concludes the thesis recapping the experimental results and future works that can be done to the tool and for the future of the project.

4.1 Conclusion

Web applications can be tested with many techniques, all of them expose issues that are discussed in the present thesis. Many literatures reviews were collected to define the development of web application and four testing techniques for the web application, compare their differences, analyze their advantages and disadvantages.

Selenium WebDriver is powerful for web application testing. It gives out maintenance and fragility between web application evolution. Furthermore, these development and testing tools have many challenges.

Growing with the application is the most critical challenge that belongs to testing tools. By testing web application dolibarr(clone 5 version from GitHub), it evaluates the Selenium WebDriver testing tool.

For the test suits does not support the translation of everything that may be appealing with the application, it should be assumed that the translation tests will be incorrect if cracks were present in the original test suits.

Regarding the evolution of test code, on average, close to 10% total number of rows changed between successive releases The same project belongs to the code associated with the selected test architecture. If the percentage is considered low, inevitable code confusion during the development of the

application, Testing must adapt to changing requirements or any kind of change AUT.

4.2 Future Work

Future works would cover the missing element and would do more by validating. Tests require more maintenance during the normal evolution of a web application.

The common project would simplify the creation of test cases devices starting from one of them, it should seek a correlation between LOCs or modifier classes, for example, TLR, MTLR MRTL, TMR metrics are recommended in this paper) and developers' effort or apply better predictions as recommended literature, try to measure how is the test code related to the application.

Besides, dynamic evaluations can be carried out to quantify the extent of the unfamiliar test code maintained by the council.

Developers do not undertake maintenance projects and assess how developers deal with the aging test code.

The fragility of the tests can be assessed by two metrics based on the number of classes and methods changed.

Based on this evaluation, it is possible to plan soon.

Provide the definition of separation cause classification by using a theoretical method based on Git description of various documents that will help to avoid these documents.

Finally, adapt automatically to an equipment development application or at least changed test case signs that developers are starting to weaken specific test types.

Check for test cases that are hidden or unchanged. It makes the piece unusable and still works. Even if they are innovative or still exist No adjustments made.

In another database study for open source projects different testing frameworks or types of different platforms and private source commercial applications It is also designed.

There may be some improvements to this study. It can be updated cross-platform test automation tools that do not have compatibility problems as it appears in this study.

Additionally good and compatible guidelines can be developed for helping developers to avoid common issues.

Furthermore, the efforts from researchers in the area of web application

testing should be focused on finding ways to overcome compatibility, layout and widget specification for Web View, documentation, problems which are described in the taxonomy of challenges.

Bibliography

[1] A Model-Based Approach for Testing the Performance of Web Applications Mahnaz Shams Diwakar Krishnamurthy Behrouz Far.

[2] Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution.

[3] M. Utting, B. Legeard. Practical Model-based Testing: A tools approach. Morgan Kaufmann, 2007.

[4] A MODEL BASED TESTING TECHNIQUE TO TEST WEB APPLICATIONS USING STATECHARTS Hassan Reza, Kirk Ogaard, Amarnath Malge.

[5] Conceptualization and Evaluation of Component-based Testing Unied with Visual GUI Testing: an Empirical Study.

[6] Mobile GUI Testing Fragility: A Study on Open-Source Android Applications Riccardo Coppola , Maurizio Morisio , and Marco Torchiano.

[7] Scripted GUI Testing of Android Apps:A Study on Diffusion, Evolution and Fragility.Riccardo Coppola, Maurizio Morisio and Marco Torchiano.

[8] Prevalence and Maintenance of Automated Functional Tests for Web Applications. Laurent Christophe, Reinout Stevens, Coen De Roover, Wolfgang De Meuter. Software Languages Lab, Vrije Universiteit Brussel, Brussels, Belgium.

[9] Implementation and evaluation of a tool for translating Visual to Layout-based android tests.Simona Saitta.

[10] WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC S. Balaji, M. Sundararajan Murugaiyan.

[11] Software fault interactions and implications for software testing DR Kuhn, DR Wallace, AM Gallo, IEEE Transactions on Software Engineering (Volume: 30 , Issue: 6 , June 2004).

[12] A. Bruns, A. Kornstadt and D. Wichmann, "Web Application Tests with Selenium," in IEEE Software, vol. 26, no. 5, pp. 88-91, Sept.-Oct. 2009, doi: 10.1109/MS.2009.144.

[13] A. Holmes and M. Kellogg, "Automating functional tests using Selenium," AGILE 2006 (AGILE'06), Minneapolis, MN, 2006, pp. 6 pp.-275, doi: 10.1109/AGILE.2006.19.

[14] Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry.Emil Borjesson ; Robert Feldt.

[15] Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. E Alégroth, R Feldt, P Kolström - Information and Software Technology, 2016 - Elsevier.

[16] GUI testing using computer vision. Tsung-Hsiang Chang,Tom Yeh,Robert C. Miller.

[17] Sikuli: using GUI screenshots for search and automation. Tom Yeh, Tsung-Hsiang Chang, Robert C. Miller.

[18] O. Zeitouni, J. Ziv and N. Merhav, "When is the generalized likelihood ratio test optimal?," in IEEE Transactions on Information Theory, vol. 38, no. 5, pp. 1597-1602, Sept. 1992, doi: 10.1109/18.149515.

[19] Analysis and Design of Selenium WebDriver Automation Testing Framework.Satish Gojare,Rahul Joshi.

[20] TestingWeb applications by modeling with FSMs Anneliese A. Andrews, Jeff Offutt, Roger T. Alexander.

[21] Lei Xu and Baowen Xu, "A framework for Web applications testing," 2004 International Conference on Cyberworlds, Tokyo, Japan, 2004, pp. 300-305, doi: 10.1109/CW.2004.7.

[22] A taxonomy of model-based testing approaches. Mark Utting, Alexander Pretschner , Bruno Legeard.

[23] A New Heuristic Measure of Fragility and Tail Risks: Application to Stress Testing.Nassim N. Taleb, Elie Canetti, Tidiane Kinda, Elena Loukoianova, and Christian Schmieder

[24] Method and system for website maintenance.Arlyn Asch,Ronald Linyard,Arie Trouw,Mark Wineman,Jonas Salling,Brian Sparks.

[25] P. Mohagheghi, B. Anda and R. Conradi, "Effort estimation of use cases for incremental large-scale software development," Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., Saint Louis, MO, USA, 2005, pp. 303-311, doi: 10.1109/ICSE.2005.1553573.

[26] I. Ku, Y. Lu and M. Gerla, "Software-Defined Mobile Cloud: Architecture, services and use cases," 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, 2014, pp. 1-6, doi: 10.1109/IWCMC.2014.6906323.

[27] M. Leotta, D. Clerissi, F. Ricca and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, 2013, pp. 272-281, doi: 10.1109/WCRE.2013.6671302.

[28] W. Hargassner, T. Hofer, C. Klammer, J. Pichler and G. Reisinger, "A Script-Based Testbed for Mobile Software Frameworks," 2008 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, 2008, pp. 448-457, doi: 10.1109/ICST.2008.51.

[29] Qing Xie, Mark Grechanik and Chen Fu, "REST: A tool for reducing effort in script-based testing," 2008 IEEE International Conference on Software Maintenance, Beijing, 2008, pp. 468-469, doi: 10.1109/ICSM.2008.4658108.

[30] D. Kung, "An agent-based framework for testing Web applications," Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004., Hong Kong, 2004, pp. 174-177 vol.2, doi: 10.1109/CMPSAC.2004.1342704.