POLITECNICO DI TORINO

Master's degree course in Computer Engineering

Master's Degree Thesis

Test Fragility: An exploratory assessment study on an Open-Source Web Application

Supervisors
Assistant Prof. Luca Ardito
Research Assistant. Riccardo Coppola
Prof. Morisio Maurizio

Candidate
Huang SHIJIE
Student Number: s233098

ACADEMIC YEAR 2019-2020

Abstract

Context: With the explosive growth of web applications in the last two decades, web application testing is an integral part of the web application development process. Frequent web application testing can minimize the chance of bugs ruining the customer experience and it also gives you a better overall idea about how your app performs, what its strengths are, and where the weak points are hidden. Automating web application testing is a highly automated process for testing web applications, and manual testing is not suitable for critical and complex applications in terms of both human resources and time, so automation testing has been introduced to overcome manual testing problems. Automation is a must in the interests of effectiveness and efficiency.

Goal: The objective of this thesis is to understand the automated web application testing techniques for web applications, and analyze the main fragility (i.e., need for maintenance of existing cases) causes of web application tests. This evaluation has been performed by means of an exploratory experiment. The test fragility results are evaluated for two different automated testing technologies.

Method: This article makes an exploratory assessment of web applications and their related vulnerability causes. Firstly a small test suite (five test cases) has been analyzed - four automated web application testing tools (SeleniumIDE-is open-source without writing code and allows for record and replay; Selenium-is an open-source framework and allows for writing code or scripts; TestComplete-automation testing tool with a hybrid object and visual recognition engine, and has script or scriptless flexibility; EyeAutomate-visual GUI Testing with intelligent image recognition technique) are applied - for Polito web application. Then introduced the concept of vulnerability in the computer field. Next, Selenium WebDriver and eyeAutomate do testing on the Dolibarr web application - The iterative process was used under 19 test cases in a test suite, then the cause of all test corruption is recorded. Moreover, the experimental results record (compute the percentage of modified tests under a test suite of one version and compute the percentage of each modified test case among the 8 versions) of Dolibarr under the Selenium/EyeAutomate test are summarized.

Results: Throughout the exploratory research, by comparing the modified LOC/instruction, the advantages and disadvantages of the two test tools for Dolibarr were compared and analyzed. The data gathered on the application Dolibarr in the test report largely show that applying the Selenium

automated testing tool is much more stable than EyeAutomate. Moreover, the summary of the main fragility cause of Dolibarr under Selenium testing gives that the most frequent causes of fragility were changes in the DOM Tree.

Conclusion: According to the results of these experiments, it is evident that fragility happens frequently on a typical open-source web application project, and that the changes in the DOM structure or graphic changes may invalidate many test cases during the evolution of a web app. These results can be used by developers and practitioners to understand which modifications can lead to much-required maintenance effort in their test cases.

Acknowledgements

I would like to thank Assistant Prof. Luca Ardito Research Assistant. Riccardo Coppola and Prof. Morisio Maurizio, who advised the project. It makes me discover the universe of web application testing research and dedicates me all the time I needed and answers all of my questions and concerns. I am extremely thankful to Research Assistant. Riccardo Coppola, who managed and administrated my research activities. I am extremely thankful to Assistant Prof. Luca Ardito and Prof. Morisio Maurizio, for supporting me well throughout my research work. I would like to thank them for guiding me throughout this journey, and I am very grateful to have received this opportunity. Their advice, discussion, and effective comments were always a source of motivation. This work would not have been possible without them.

I would like to thank my parents for encouraging me, motivating me and believe in me, this thesis would not have been possible without them. I would not have reached this stage without the support, love, and prayers of my family and friends.

Abbreviations

CSRF	Cross-Site Request Forgery
API	Application Program Interface
GUI	Graphical User Interface
BDD	Behavior-Driven Development
AI	Artificial Intelligence
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
QA	Quality Assurance
URL	Uniform Resource Locator
DLL	Dynamic-Link Library
XML	Extensible Markup Language
ERP	Enterprise Resource Planning
CRM	Customer Relationship Management
DOM	Document Object Model
LOC	Lines Of Code

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the explosive growth of web applications in the last two decades, web application testing is an integral part of the web application development process. We know testing web applications can be more challenging than testing traditional software. Software testing has been an effective approach to ensuring the quality of web applications, [1, 2].

If the web app development is done with Agile principles applied, it's important to "mix" the web app testing phase with the development phase properly. It could be summarised into 6 steps.

1. Functionality Testing

   With web functional testing, it should guarantee that all clients' requirements are met and make sure that the web application is functionally correct. In detail, Functionality testing checks the database connection, all links in the web pages, cookies, forms used for submitting and/or getting info from the user, etc. Once you perform a set of tasks whatever automatically or manually to test a web app, you should compare results with the expected output. This should be done several times with different data input to make sure the level of accuracy, then it could consider the web app is functionally correct. It should be done early in the developing stages to speed up the whole app-building process and it reduces risks toward the end of the cycle.

2. Usability testing

   Usability testing is about how to test the website, includes test the navigation and controls, Content checking, Check for user intuition. Normally, It is divided into 4 phases to complete this testing:

(a) It develops a test strategy by examining all functions of the web application (including navigation, content).

(b) Recruiting the internal or external test participants (External testers can do usability testing through simulating your expected user base, or internal testers can do the testing by the developers themselves).

(c) The team of experts runs usability testing.

(d) Analyzing the results and doing web app adjustment. It helps to overcome the usability issues then make the web app better and eyes-catching.

3. Interface testing

   Interface testing not only verifies all interaction between the app server and the webserver run correctly, but also care about displaying of error messages, and it also is used to determine whether the interruptions by the server and/or by the user are handled properly.

4. Compatibility testing

   Compatibility testing should guarantee Browser compatibility such as Chrome, Internet Explorer, Safari, Firefox, Operating system compatibility such as Windows, Mac, Linux, and is compatible with various devices like a notebook, mobile, etc.

5. Performance testing

   Normally, Checking performances to verify the responsiveness and stability of the web apps are under various load conditions.Performance testing includes testing under different internet speeds such as under normal speed, peak speeds. There are some performance testing types as described below.

   Load testing: It measures business-critical transactions and monitors the load on the database, application server, etc.

   Stress testing: It discovers the web app's breaking point and determines how the web app recovers if the current load goes well above the upper limit capacity.

   Soak testing: It's also called endurance testing. It tests the system parameters under continuous expected load then determine the relatively good one.

Spike testing: It suddenly increases the number of users by a very large amount and measures the performance of the system. It aims at whether the system can sustain the workload.

6. Security testing

   The last step verifies web app security such as data theft, unauthorized access, and other malicious actions. This testing will point out the web app's weak points, one way is normally applied which is serious of fabricated malicious attacks to test how the app response performs under these situations. Once a security issue is detected, testers try their best to overcome them.

Here some of the techniques to verify the security level of the web application are listed: Injection, Broken Authentication and Session Management, Cross-Site Scripting, Insecure Direct Object References, Security Misconfiguration, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery (CSRF), Using Components with Known Vulnerabilities, Unvalidated Redirects and Forwards.

So when the application testing runs smoothly, the web app is ready to be released. Under software testing, Manual testing is not suitable for critical and complex applications in terms of both human resources and time. So it is not a practical choice anymore. So Automation testing has been introduced to overcome manual testing problems. Automating the execution phase of the software testing cycle is particularly the most popular way in the automation field and No human intervention required, it improves the effectiveness, efficiency, and coverage of software testing. Automated testing includes functional automated testing and automated performance testing.

The most efficient way to perform test automation for web applications is to adopt a pyramid test strategy. This pyramid test strategy includes three different levels of automated testing. Unit testing represents the cardinality and maximum percentage of this test automation pyramid. Next is the service layer or API testing. Finally, GUI testing is at the top. But Automated web application testing offers its own unique set of challenges nowadays. One of the Challenges is that automatically test on multiple Browsers (and possibly multiple versions of each Browser); test on different devices/form factors (where mobile views may differ from standard desktop views); be able to reuse test cases, or partial test scenarios, etc. Hence, tool support for web test automation exists necessarily. We will take a look at some of the web application testing tools such as Zephyr, TestLink, PractiTest, TestComplete,

Selenium, HP UFT, Tricentis, BugZilla, etc. For example, to facilitate easy test case creation without writing code, a popular option is to employ tools that allow for record and replay such as Selenium IDE, [3], which is free and open-source. In cases where writing code or scripts, while still leveraging some external tooling, is a viable option, several open-source frameworks also exist such as Selenium WebDriver, [4]. Moreover, TestComplete, claim that the Easiest-to-Use Automated UI Testing Tool with Artificial Intelligence leverages the industry's first automation testing tool with a hybrid object and visual recognition engine to test every desktop, web, and mobile application with native BDD-style Gherkin syntax and script or scriptless flexibility. EyeStore has tools for visual GUI Testing, Automate any app on any platform using intelligent image recognition. This is possible due to a combination of image recognition and AI. In this report, those 4 automated web application testing tools that I did research will be introduced in the next chapters in detail.

The remainder of this report is organized as follows: The background section presents the rationale of the web application, testing, and implementing some basic tests on the web application: https://www.polito.it. It uses 4 different automated web application testing tools and introduces those open source automation tools. The experiment section discusses the fragility concept, explains our testing procedure on an open-source web application: Dolibarr using Selenium WebDriver and EyeAutomate technique, and the definition of test cases for this experiment. The results section reports the result of the experiment section. In the last chapter, the conclusion has been provided for the thesis.

Chapter 2

Background

2.1 Web applications

Web applications (such as online auction, webmail, word processors, online forms, spreadsheets, file conversion, etc) are ubiquitous in our life. The web application is a client-server computer application program that is stored on a remote server and delivered over the internet through a browser interface. It means users can access the web application through a network on a web browser such as Firefox, Safari or Google Chrome. Therefore users don't need to install web applications anymore. It is friendly to the business and end-user, and it can run on multiple platforms regardless of operating systems or computers, but a compatible browser must be guaranteed.

Most web applications are written in JavaScript, HTML5, or Cascading Style Sheets (CSS). Client-side programming executed on the browser builds front-end to present information to users by using those above languages, Server-side programming executed on the server typically utilizes Python, java, Ruby language to create scripts (PHP, ASP, JSP, etc) which a web application will use to handle the storage and retrieval of the information.

A web application works as below.

1. The user triggers a request to the webserver over the Internet, either through a web browser or the application's user interface. It is called a client-side request.

2. Web servers manage client-side requests, so web servers will forward the client-side request to the web application server.

3. The web application server performs the requested task such as querying

the database or processing the data, then generates the result of the requested data as a response to the webserver.

4. The web server transfers the response to the client, then the response shows on the browser to the user.

2.2 Testing

According to the ANSI/IEEE 1059 standard, Testing means that evaluating the features of a system or its component (s) and detect any defects, errors, or missing requirements contrary to the actual requirements. Testing can be done at all stages of module development: requirements analysis, interface design, algorithm design, implementation, and integration with other modules. Testing is performed by the testing team, after that, the testing team will report to the developer team to debug.

Testing is so important. When the Quality of the product is user-friendly, powerful, and useful, customers will have confidence in the team and the organization. If software results in failures, It would be very expensive to fix in the future or the later stages of the development even worse is required to redesign. Testing helps to identify web applications weak points and increase app work for several times. Testing is necessary because it can point out the defects and errors that were made during the development phases. Ignoring testing as a critical part of the success of your project will only lead to failed projects and significantly impactful issues in production. We should pay more attention to it.

Firstly, we need to know different testing type roles depended on the project and the organization, such as QA manager who has to control and manage the tests, QA Analyst with a profile who usually manage to design the tests, tester who is in charge of executing test cases, Professionals with more technical profiles who can automate tests or performance tests.

Before going further into details of software testing, here few terms related to software testing are being explained.

2.2.1 Test plan

“A test plan is an artifact that expresses the objectives, scope, technique, approach, and focus of a software testing effort”, [5]. Typically the Quality Assurance Team Lead will write a Test Plan. A test plan includes:

1. Introduction to the Test Plan document

2. Assumptions when testing the application
3. Test cases list
4. List of features to be tested
5. When you test software, you should think about what sort of approach to use
6. The list of deliverables that need to be tested
7. What resources are allocated for testing
8. Risks during the testing process
9. A schedule of tasks and milestones to be achieved

2.2.2 Test case

"A test case is an artifact that delineates the input, action and expected output corresponding to that input", [6]. A fundamental part of software testing is a test-case generation. There are many types of test cases such as functional, negative, error, logical test cases, physical test cases, UI test cases, etc. Test cases are used to expose faults within the web app and they are written to keep track of the testing coverage of a software. Furthermore, creating more test cases requires more resources.

2.2.3 Test script

"A test script is a combination of test cases to test a particular function or component of the system", [6].

2.2.4 Test coverage

Testing could be endless, and we use coverage in which we implement a program to determine how thoroughly a test suite exercises it. Test coverage is a technique to measures the amount of testing performed by a set of tests. It could determine all the decision points and paths used in the application, then create additional test cases to increase coverage, it also can find a requirement area but not implemented by a set of test cases. Test coverage helps us identify meaningless test cases that don't increase coverage and a

quantitative measure of test coverage which is an indirect method for quality check. In the meantime, test coverage has some disadvantages. Most of the task in test coverage is no way to automate, then we need more effort to analyze the requirements and create test cases. Even you can count test coverage features and measure against several tests, but judgment error still could happen.

2.2.5 Black-Box Testing

The black-box testing technique is based directly on specified functional requirements and has no concern considering the ultimate program structure, [9]. Black box testing is also known as data-driven testing, input/output-driven testing, [7], or requirements-based testing, [8]. As in black box testing, no more than the functionality of the software module is of use, it is also referred to as functional testing - a testing method emphasizing to execute the functions and examine their input and output data, [9]. The tester doesn't take into account the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

2.2.6 White-Box Testing

White Box Testing can also be termed as glass box testing, clear box testing, and structural testing,[7, 8]. Contrary to black-box testing, The tester needs to know the source code and find out which unit/chunk of the code is behaving inappropriately. It helps in optimizing the source code.

2.2.7 Grey-Box Testing

Grey-Box Testing seems a mix of Black Box and White Box and offers combined benefits of black-box and white-box testing wherever possible. Unlike black-box testing, where the tester only tests the application's user interface; for grey-box testing, the tester has permission to design documents and the database. Grey-Box Testing can take the ease-of-use, straightforward approach of Black Box testing and leverage it against the in-depth, code targeted testing of White Box. Grey-Box Testing is done from the point of view of the user and not the designer.

2.2.8 Manual testing

Manual testing doesn't use any automated tool or any script. The tester tests the software to identify any unexpected behavior or bug. Testers use test plans, test cases, or test scenarios to test software, this ensures the completeness of testing. There are many stages for Manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

2.2.9 Automation testing

Automation of software testing is the process of creating a program (test script) that simulates the manual test case steps in whatever programming/scripting language, [10, 11] with the help of other external automation helper tool, [12, 13]. Automation can reduce time and costs. Automation is generally supportive while managing recurring responsibilities like unit testing and regression testing, where test cases are carried out whenever modifications are completed, [14]. In contrast to manual testing, automated testing is inappropriate for tasks in which there is little repetition, [15], such as explorative testing or late development verification testing. Manual testing is more suitable for these activities as building automation is an extensive task and feasible only if the case is repeated several times, [15]. Overall, the main drawbacks of testing automation are the costs such as implementation costs, maintenance costs, and training costs. Implementation costs include direct investment costs, time, and human resources. The correlation between these tests automation costs and the effectiveness of the infrastructures have been discussed in literature, [16].

The following tools can be used for automation testing: [1], HP QuickTest Professional [2], Selenium IDE [3], EyeAutomate [4], IBM Rational Functional

[1] https://selenium.dev/

[2] https://www.tutorialspoint.com/qtp/index.htm

[3] https://selenium.dev/selenium-ide/

[4] https://eyeautomate.com/eyeautomate/

Tester [5], SilkTest [6], TestComplete [7], LoadRunner [8], Visual Studio Test Professional [9], WATIR [10].

I will introduce some of them later.

2.3 Exploration of web-based testing techniques

At first, an exploration of the different testing techniques that are available for web-apps has been conducted. As the software object for the exploration, the official web app of Polito has been selected. In Figure 2.1-2.5, 5 screen captures of the web app are shown. This app allows students or professor/researchers to log in and access a set of specific features: your personal data, lectures timetable, info about mailbox, info about education resources (include course videos online), info about language studying resources, info about jobs, info about tuition fees and certificates, info about thesis topics, info about the career and the possibility of booking exam calls, etc.

[5] https://www.ibm.com/us-en/marketplace/rational-functional-tester

[6] https://www.microfocus.com/en-us/products/silk-test/overview

[7] https://smartbear.com/product/testcomplete/free-trial/

[8] https://www.microfocus.com/en-us/products/loadrunner-professional/overview

[9] https://visualstudio.microsoft.com/vs/test-professional/

[10] http://watir.com/

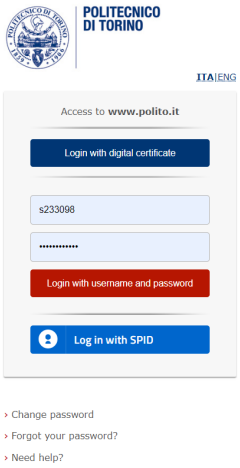


Figure 2.1. Login

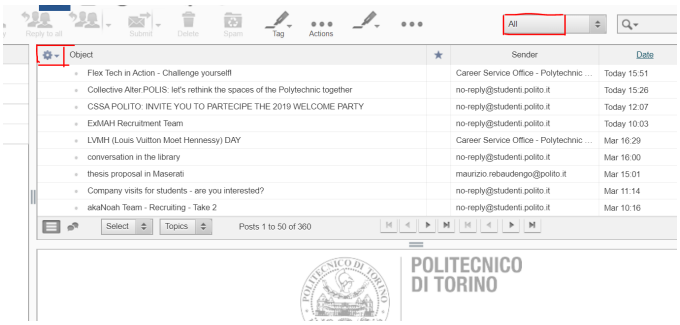


Figure 2.2. Drop-Down List

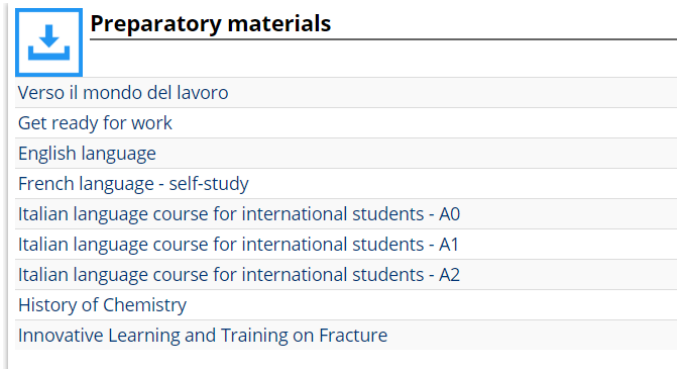


Figure 2.3. Operate Video

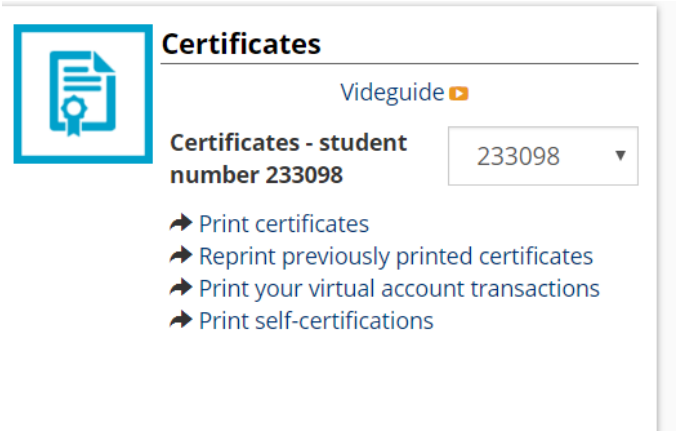


Figure 2.4. Print Certificate

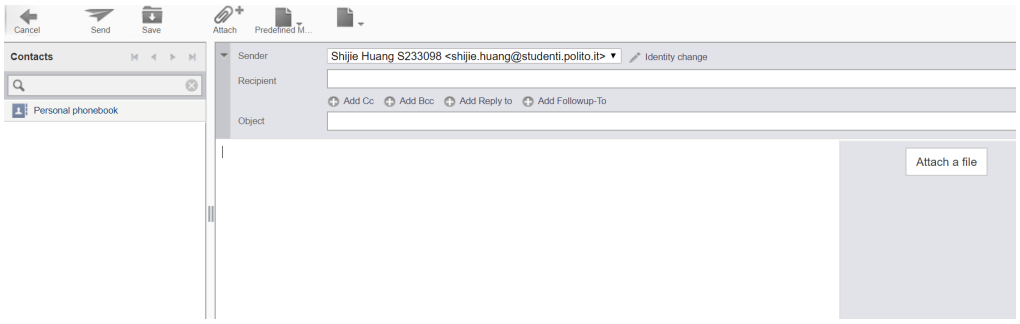


Figure 2.5. Write Email

4 automated web app testing tools (selenium, selenium IDE, TestComplete, EyeAutomate) were used to develop test suites for the Polito web application. The main usage scenarios of the app were exercised with five different test cases, listed in Table 2.1.

Table 2.1. Description of Usage Scenarios

Name	Description
T1	Student login
T2	Check list in email
T3	Check video
T4	Check certificate print
T5	Check email writing

For more details please see the next section.

2.4 Selenium IDE

Selenium IDE is a Chrome and Firefox add-on as an open-source record and playback test automation tool for the web. It helps generate and maintain site automation, tests, and achieve the goal which removes the need to manually step through repetitive operations. Once records finish, generated recorded test scripts can be exported to C#, Java, and Ruby or Python programming languages. Then those exported scripts can be used by Selenium RC (Deprecated) or Selenium WebDriver.

2.4.1 SetUp

Setting up selenium IDE means that installing the extension on the browser.

1. Open Chrome or Firefox web store on the web browser
2. Search Selenium IDE, then click the button to add it into Chrome/Firefox
3. Close and restart the Chrome/Firefox browser
4. Launch Selenium IDE by clicking its icon from the menu bar in the browser
5. Upon launching the Selenium IDE will be presented with a welcome dialog

2.4.2 How it works

Once I launch Selenium IDE, Firstly, According to the demands to select one option in the welcome dialog. For the first time to use, I select the first one. After that name this new project and fill one base URL which is https://www.polito.it/. As mentioned above, a new browser window will open, load the base URL, and start recording. Next, Selenium IDE interacts with the web page and each of the actions will be recorded. Finally, Stopping recording with the red icon, so I need to switch to the IDE window, then steps of operation using selenium IDE commands are shown.

2.4.3 Example test suite with SeleniumIDE

Each SeleniumIDE command line is composed of a tuple: <Command, Target, Value>. Command indicates either an event (such as click, select, type) that is performed on the user interface during the recording process or action specific to Selenium's control of the replay process. When the user interacts with interface elements (input fields, drop-down lists, etc), Target indicates the interface elements during the recording process. A value indicates that the user types a value in the specified locator, such as the value that the user selected in the drop-down list or the value typed in the text field.

https://www.polito.it

	Command	Target	Value
7	*click*	id=j_password	
8	*type*	id=j_password	zzsjdwbhsj123
9	*click*	id=usernamepassword	
10	*click*	id=j_username	
11	*type*	id=j_username	s233098
12	*click*	css=.content	
13	*click*	id=j_password	
14	*type*	id=j_password	ksxdzchsj123
15	*click*	id=usernamepassword	
16	*click*	css=.truncate-login	
17	*click*	linkText=Logout	
18	*click*	css=.dropdown-toggle > .hidden-xs	

Figure 2.6. Login with Selenium IDE

Figure 2.6 shows the login process. It clearly shows that a click operation is performed on the "id=j_password" of the HTML element, and then the typing operation is performed on the same id. Furthermore, there also is the usage of CSS and linkText in Target to point out the positions of the elements.

https://www.polito.it

	Command	Target	Value
5	click	css=.wrapper	
6	click	id=j_password	
7	type	id=j_password	ksxdzchsj123
8	click	id=usernamepassword	
9	click	linkText=Portale della Didattica	
10	click	linkText=Posta	
11	click	id=rcmlistfilter	
12	select	id=rcmlistfilter	label=Cancellato
13	click	id=rcmlistfilter	
14	click	id=listmenulink	
15	click	css=.floating:nth-child(2) li:nth-child(4) input	
16	click	id=listmenusave	

Figure 2.7. Drop-down List with Selenium IDE

In Figure 2.7, Line 12 does a select operation for the target element which id is “rcmlistfilter”, and executes to select the label is “Cancellato”.

9	click	linkText=Portale della Didattica
10	click	linkText=Corso di lingua italiana per stranieri - A0
11	click	linkText=2018 Pillola 005
12	click	css=.video-js-box
13	mouse over	id=btnPlay
14	click	id=btnPlay
15	mouse over	id=btnTag
16	mouse out	id=btnTag
17	mouse over	id=video62072
18	click	id=video62072

Figure 2.8. Operate Video with Selenium IDE

In Figure 2.8, Line 13 simulates a user hovering a mouse over the element indicating by id=btnPlay. Line 16 simulates the user moving the mouse pointer away from the element with id=video62072.

9	*click*	linkText=Portale della Didattica	
10	*click*	linkText=Segreteria online	
11	*click*	linkText=Stampa autocertificazioni	
12	*select window*	handle=${win7432}	
13	*click*	id=blocco301	
14	*click*	css=.panel-heading-polito	
15	*choose ok on next confirmation*		
16	*click*	id=btnprt	
17	*assert confirmation*	ATTENZIONE, se si procede con la stampa il documento no n sara piu modificabile	
18	*webdriver choose ok on visible confirmation*		

Figure 2.9. Print Certificate with Selenium IDE

Select window: Selects a popup window using a window locator. Once a pop-up window has been selected, all commands will go to that window. Window locators use handles to select windows. Therefore, line 12 shows the selection of a window that is represented by a variable window handle. Line 15 chooses OK on the next confirmation affects the next confirmation alert. This command will accept Line 17 assert confirmation and Confirm that a confirmation has been rendered. Line 18 WebDriver chooses OK on visible confirmation: Affects a currently showing confirmation alert. This command instructs Selenium to accept it. So the certificate will be printed successfully.

9	*click*	linkText=Webmail	
10	*wait for element not present*	linkText=Webmail	20
11	*click*	id=rcmbtn111	
12	*click*	id=_to	
13	*type*	id=_to	1372937384@qq.com
14	*click*	id=bcc-link	
15	*click*	css=#compose-bcc label	
16	*type*	id=_bcc	s233098@studenti.polito.it
17	*click*	id=compose-subject	
18	*type*	id=compose-subject	welcome to italy
19	*click*	id=composebody	
20	*type*	id=composebody	hi,everybody!.....--------

Figure 2.10. Write Email1 with Selenium IDE

20	type	id=composebody	hi,everybody!.....--------
21	click	id=rcmbtn121	
22	type	name=_attachments[]	
23	click	id=rcmbtn126	
24	click	id=rcmbtn110	

Figure 2.11. Write Email2 with Selenium IDE

For write emails procedure (see Figure 2.10, Figure 2.11), we can see the selenium commands work on elements identified by id. Line 10 wait for element not present: Wait for a target element to not be present on the page. The target element is identified by linkText=Webmail, It is designed to disappear from web pages containing webmail elements and display the next page after 20 milliseconds.

2.5 Selenium

In selenium WebDriver, robust, browser-based regression automation suites and tests and scale and distribute scripts across many environments could be created. In detail, Selenium WebDriver could create and execute test cases because it provides a programming interface. Test scripts are written to identify web elements on a web page and then perform the required actions on those elements.

Selenium WebDriver can drive a browser natively either locally or on remote machines, it depends on how you use it. Selenium WebDriver supports Internet Explorer, Mozilla Firefox, Google Chrome, Safari, and supports many programming languages to write tests, such that java, C#, JavaScript, PHP, Ruby, Pearl, Python, etc. In this report, I will use java programming. It is especially important to pay attention to detail about the driver, WebDriver has a built-in Firefox driver (Gecko Driver) implementation. For other browsers, you need to insert their browser-specific drivers to communicate and run tests. A browser-specific driver is for creating a secure connection between WebDriver and browser. You will see its specific operation in the next content. Moreover, Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

2.5.1 SetUp

To install Selenium means to set up a project in development so write a program using Selenium.

Here I use java language as an example to describe the whole procedure of work.

There are 4 steps to setup WebDriver.

1. Download and install java 8 or higher version

   Download the newest JDK version with this link: http://www.oracle.com/technetwork/java/javase/downloads/index.html, then set path or environment variable.

2. Download and configure Eclipse IDE or choose other Java IDE

   Open this website: https://www.eclipse.org/downloads/, then find and download "Eclipse IDE for Java Developers", choose one according to my current operating system version.

3. Download Selenium WebDriver Java client

   Open this website: https://docs.seleniumhq.org/download/, find "Selenium Client&WebDriver Language Bindings" and then choose download link about java client driver.

4. Configure selenium web driver

   (a) Launch Eclipse IDE

   (b) Create a new java project named Demotest (Demotest is as a test suite)

   (c) Create a new class named First (First.java) under src folder

   (d) Right-click Demotest folder, then select "properties". After opening the property of Demo test window, choose "Java Build Path" property, switch to "Libraries" option at the right panel, then click "Add External JARs" button, then find all selenium jar files we download at step 3, finally click "Apply and Close" button.

Now it configures successfully.

2.5.2 How it works

Here it uses java language as an example to describe the whole procedure of work. Here the first selenium automated test script is created. It aims to implement those actions below:

1. Call Chrome browser
2. Open www.google.com
3. Click Google search text box
4. Type keywords: politecnico di torino
5. Press "Enter" key

Here a test case is created step by step to understand each component.

1. Launch Eclipse IDE and open the Demo test project created in the last section. Then start to compile First.java. Firstly, Instantiating a driver object:

   WebDriver driver = new ChromeDriver ();

   Pay attention: ChromeDriver is maintained/supported by the Chromium project itself. WebDriver works with Chrome through the Chromedriver binary (found on the Chromium project's download page). You need to have both Chromedriver and a version of the chrome browser installed. Chromedriver needs to be placed somewhere on your system's path for WebDriver to automatically discover it. The Chrome browser itself is discovered by Chromedriver in the default installation path. These both can be overridden by environment variables.

2. Then, fetching www.google.com web page

   driver.get ("https://google.com/");

3. Next, You need to add unique identifiers to network elements such as Google search text boxes to automate the logo through test scripts. These unique identifiers are configured with some commands/grammar to form a locator. Enables the locator to locate and identify specific network elements in the context of the web application.

   The method for finding a unique identifier element involves checking the HTML code. The Java syntax for locating elements by "name" in Selenium WebDriver is given below.

driver.findElement (By.name (<element ID>));

so do

WebElement element = driver.findElement (By.name("q"));

4. Now try to type the keyword that you want to search, for example, Politecnico di Torino.

   element.sendKeys ("politecnico di torino");

5. Finally, submit our request.

   element.submit ();

The complete sample code is as follows:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
public class First {
  public static void main (String[] args) {
String projectlocation=System.getProperty ("user.dir");
System.setProperty ("webdriver.chrome.driver",projectlocation+"/lib/web
drivers/chromedriver.exe");
WebDriver driver=new ChromeDriver ();
driver.get ("https://google.com/");
WebElement element = driver.findElement (By.name ("q"));
element.sendKeys ("politecnico di torino");
element.submit ();
}
}
```

Until now, I have created the first test script for implementing selenium automation. I can verify it by running it in Eclipse IDE now.

2.5.3 Example test suite with Selenium

Figure 2.1 depicts the user interface in the Polito web app used to log in and contains (from top to bottom) text fields, a submit button. This test script has at least 20 lines of code. Selenium web driver test script screen capture for this web page please see Figure 2.12.

```
String projectlocation=System.getProperty("user.dir");
System.setProperty("webdriver.chrome.driver",projectlocation+"/lib/webdrivers/chromedriver.exe ");
   WebDriver driver = new ChromeDriver();
   driver.get("https://www.polito.it/");
   driver.manage().window().setSize(new Dimension(1296, 696));
   driver.findElement(By.cssSelector(".dropdown-toggle > .hidden-xs")).click();
   driver.findElement(By.id("j_username")).click();
   driver.findElement(By.id("j_username")).sendKeys("s233098");
   driver.findElement(By.cssSelector(".content")).click();
   driver.findElement(By.id("j_password")).click();
   driver.findElement(By.id("j_password")).sendKeys("ksxdzchsj123");
   driver.findElement(By.id("usernamepassword")).click();
   driver.findElement(By.id("j_username")).click();
   driver.findElement(By.id("j_username")).sendKeys("s233098");
   driver.findElement(By.cssSelector(".content")).click();
   driver.findElement(By.id("j_password")).click();
   driver.findElement(By.id("j_password")).sendKeys("zzsjdwbhsj123456");
   driver.findElement(By.id("usernamepassword")).click();
   driver.findElement(By.cssSelector(".truncate-login")).click();
   driver.findElement(By.linkText("Logout")).click();
   driver.findElement(By.cssSelector(".dropdown-toggle > .hidden-xs")).click();
```

Figure 2.12. Login with Selenium

For username and password and submit button elements can be identified by id locator which belongs to attribute-based locators, login element is identified by CSS selector which belongs to hierarchy-based locators, logout element is detected through its textual description.

Figure 2.2 shows the use of a drop-down list in an email web page. This test script has at least 25 lines of code. Selenium WebDriver test script screen capture for this web page please see Figure 2.13.

```
driver.findElement(By.linkText("Portale della Didattica")).click();
 for(int i=0;i<2;i++) {
     try {driver.findElement(By.partialLinkText("Posta")).click();
       break;
       }catch(Exception e) {
           System.out.println(e.getMessage());
       }
 }
driver.findElement(By.id("rcmlistfilter")).click();
  WebElement dropdown = driver.findElement(By.id("rcmlistfilter"));
  dropdown.findElement(By.xpath("//option[. = 'Cancellato']")).click();
driver.findElement(By.id("rcmlistfilter")).click();
driver.findElement(By.id("listmenulink")).click();
driver.findElement(By.cssSelector(".floating:nth-child(2) li:nth-child(4) input")).click();
driver.findElement(By.id("listmenusave")).click();
```

Figure 2.13. Drop-down List with Selenium

Firstly, clicking “Posta” in the navigation of the main page to go to the mailbox, here I used for a loop because of loading times, besides, X of element text “Posta X” will change with the arrival of new mail, so use partialLinkText locator to search “Posta X”. Second, The drop-down list is identified

by id locator, use Xpath locator with content “Cancellato” when select one option from the drop-down list. Third, The other drop-down list also is identified by id locator, use a CSS selector to locate an option. Finally, the save button is identified by the id locator.

Figure 2.3 shows an operation process to watch the instructional video. This test script has at least 31 lines of code. Selenium WebDriver test script screen capture for this web page please see Figure 2.14.

```
driver.findElement(By.linkText("Portale della Didattica")).click();
JavascriptExecutor js =(JavascriptExecutor) driver;
js.executeScript("scrollBy(0,7000)");
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.findElement(By.linkText("Corso di lingua italiana per stranieri - A0")).click();
driver.findElement(By.linkText("2018 Pillola 005")).click();
driver.findElement(By.cssSelector(".video-js-box")).click();
  WebElement element = driver.findElement(By.id("btnPlay"));
  Actions builder =  new Actions(driver);
  builder.moveToElement(element).perform();
driver.findElement(By.id("btnPlay")).click();
  WebElement element1 = driver.findElement(By.id("btnTag"));
  Actions builder1 = new Actions(driver);
  builder1.moveToElement(element1).perform();
  WebElement element2 = driver.findElement(By.tagName("body"));
  Actions builder2 = new Actions(driver);
 builder2.moveToElement(element2, 0, 0).perform();
  WebElement element3 = driver.findElement(By.id("video62072"));
  Actions builder3 = new Actions(driver);
   builder3.moveToElement(element3).perform();
driver.findElement(By.id("video62072")).click();
```

Figure 2.14. Operate Video with Selenium

Once there is student main webpage, Scroll down the screen then locate the video element text “Corso di lingua italiana per stranieri - A0” using linkText locator, next select one of videos by linkText locator, Then CSS selector selects video download in the download mode. For Action class, the user-facing API for emulating complex user gestures. Using this class rather than using the Keyboard or Mouse directly.

Figure 2.4 shows an operation process to print the certificate. This test script has at least 30 lines of code. Selenium WebDriver test script screen capture for this web page please see Figure 2.15.

```
driver.findElement(By.linkText("Portale della Didattica")).click();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.findElement(By.linkText("Segreteria online")).click();
String urlparent=driver.getCurrentUrl();
String parentWinHandle=driver.getWindowHandle();
System.out.println("parent window....."+urlparent+"....."+parentWinHandle+"-----\n");
WebElement currentweb= driver.findElement(By.linkText("Stampa autocertificazioni"));
currentweb.click();
Set<String> handleslist= driver.getWindowHandles();// Get the window handles of all open windows
for(String han:handleslist) {
    if(!han.equals(parentWinHandle)) {
        driver.switchTo().window(han);
    }
}
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.findElement(By.id("blocco301")).click();
driver.findElement(By.cssSelector(".panel-heading-polito")).click();
driver.findElement(By.id("btnprt")).click();
Alert al= driver.switchTo().alert();
al.accept();
```

Figure 2.15. Print Certificate with Selenium

First, click “Segreteria online” in the navigation of the main page by linkText locator, Next linkText locator identifies “Stampa autocertificazioni”, there will be a newly launched window corresponding to a new window handle, I use the window handle to simulate windows switching operation. Next, select the content you wanna print identifying by id and Cssselector locators. Finally, a unique identifier can be used for the detection of the print button, It will pop up an alert box to confirm the print operation, here I used Alert class to implement it.

Figure 2.5 shows an operation process to write an email. This test script has at least 31 lines of code. Selenium WebDriver test script screen capture for this web page please see Figure 2.16.

```
    driver.findElement(By.linkText("Webmail")).click();
    WebDriverWait waits=new WebDriverWait(driver,20);
    WebElement waitnextpage;
    waitnextpage=waits.until(ExpectedConditions.visibilityOfElementLocated(By.id("rcmbtn111")));
    waitnextpage.click();
    driver.findElement(By.id("_to")).click();
    driver.findElement(By.id("_to")).sendKeys("1372937384@qq.com");
    driver.findElement(By.id("bcc-link")).click();
    driver.findElement(By.cssSelector("#compose-bcc label")).click();
    driver.findElement(By.id("_bcc")).sendKeys("s233098@studenti.polito.it");
    driver.findElement(By.id("compose-subject")).click();
    driver.findElement(By.id("compose-subject")).sendKeys("welcome to italy");
    driver.findElement(By.id("composebody")).click();
    driver.findElement(By.id("composebody")).sendKeys("hi,everybody!.....--------");
    driver.findElement(By.id("rcmbtn121")).click();
    driver.findElement(By.name("_attachments[]")).sendKeys("D:\\seleniumuse\\seleniutests\\src\\up.txt");
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    driver.findElement(By.id("rcmbtn125")).click();
  WebElement ad=  driver.findElement(By.id("rcmbtn110"));
     System.out.println("-------"+ad.getTagName()+"-------");
  ad.click();
```

Figure 2.16. Write Email with Selenium

First, come to the mailbox homepage, but loading time is required, here I use the WebDriver Wait class to wait for an element in the home page to be visible, then I will write a new email. For Recipient, Bcc, Object elements, and text field, unique identifiers are used in id locator. For attaching a file element, here it is identified by name locator. Finally, an id locator is used to implementing save functionality.

2.6 TestComplete

TestComplete is an automated UI testing tool with Artificial Intelligence, that makes it fast and easy to create, maintain, and execute functional tests across desktop (these applications are executed on desktop computers running the Windows operating system), web (these applications are executed in web browsers (including those web browsers that are embedded into desktop applications), and mobile (these applications are executed on Android or IOS devices) applications. With TestComplete, Scaling your automated testing efforts and maximizing test coverage. It is easy to integrate with test automation and continuous testing applications, frameworks, and controls you already use.

TestComplete provides special features for automating test actions, creating tests, defining baseline data, running tests, and logging test results. For example, it includes a special “recording tests” feature that lets you create tests visually. You just need to start recording, perform all the needed

actions against the tested application and TestComplete will automatically convert all the "recorded" actions to a test. TestComplete also includes special dialogs and wizards that help you automate comparison commands (or checkpoints) in your tests.

TestComplete supports various testing types and methodologies such as unit testing, functional and GUI testing, regression testing, distributed testing and others.

TestComplete basic features as described below:

1. Test any application using object recognition with AI

   We can accurately find dynamic UI elements Because TestComplete object recognition combines property-based and AI-powered visual recognition. And it should be quick and easy.

2. Build and maintain automated GUI tests faster than ever

   The customizable object repository has Record&Replay capabilities, so it can create automated UI tests in seconds. And the TestComplete intelligent recommendation system would help you keep up with application updates and test maintenance.

3. To script or not to script; Do both with a flexible test automation tool

   You have the flexibility to choose between scripting or scriptless testing because TestComplete has capabilities about writing script with five languages (VBScript, JavaScript, Python, DelphiScript, JScript) or you also can generate keyword-driven tests by using it.

4. Automate business requirements with native support for BDD syntax

   TestComplete has native support for Gherkin's Given-When-Then scenarios, so non-technical and technical stakeholders can quickly convert business requirement specifications written in plain English into automated tests, It is friendly to start their full-blown Behavior-Driven Development. TestComplete does not need additional plug-ins, can build and convert features files into automated UI functional tests. And an IDE, a Gherkin interpreter, a test runner, and reporting insights they are all in TestComplete.

2.6.1 SetUp

1. Firstly, download the TestComplete executable file by clicking the link below: https://smartbear.com/product/testcomplete/

free-trial/

2. Find downloaded TestComplete_xxxxx.exe in the local file directory, double-click on it, The installation will start.

3. To enable the Intelligent Quality add-on, on the next page, click the link to read the third-party license agreement. If license terms are agreed, leave the Intelligent Quality Add-on check-box enabled and click Next.

4. On the next page, select a folder path for installing TestComplete.

5. Click Install to start the installation.

6. After the installation completes, I do click operation to Start a 30-Day Trial in the dialog and wait until TestComplete activates the trial license. After the activation is over, the TestComplete working window is shown.

2.6.2 How it works

Now let's explain how to record and playback a simple web test in TestComplete. In my explanations, I will use Google search Politecnico di Torino, the same as applying other automated web application testing tool.

So the basic steps are below:

1. Launch chrome browser

2. Open www.google.com web page

3. Type "Politecnico di Torino" in Google text search box

In TestComplete, To create tests in two ways:

1. Create tests manually - It requires you have good experience in creating tests. You type all the needed commands and actions via script objects or keyword test commands. This approach is helpful when you need to create very powerful and flexible tests. However, creating tests manually could meet various problems. For example, you are familiar with the classes and names of your application objects, so it can work well.

2. Record tests - It will create tests easily. You create a test visually using the Recording function of TestComplete, It means that TestComplete will automatically recognize these actions, you record the performed actions to a script or keyword test directly. This approach does not require much experience in creating tests.

So I as a beginner will use keyword tests to understand the working principle.

The recording procedure describes as below:

1. Starting recording by selecting *Test* > *Record* > Record Keyword Test from the TestComplete main menu (It also can start recording by clicking "*RecordTest*" on the Start Page). Then TestComplete will switch to the recording mode and display the Recording toolbar (see Figure 2.17) on the screen. The toolbar is collapsed by default and shows only the most commonly used commands during the recording. Then performing additional actions during the recording by expanding the toolbar, like *Pause* or *Stop* recording, and change the type of the recorded test (keyword test, script code, or low-level procedure).

2. After starting the recording, perform the desired test actions: launch the tested application (Our example is launch Chrome browser), then open https://www.google.com web page (wait until the selected web browser starts and navigates to the main page of the google), Click in the Search text box and type Politecnico di Torino, then type enter key. Close the web browser by clicking the *X* button on its caption bar. Those actions will be recorded by TestComplete.

   Pay attention: (close Chrome browser before starting to record)

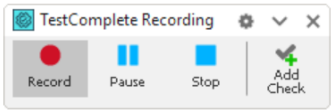


Figure 2.17. Recording Menu

3. After all the test actions are over, stop the recording by pressing *Stop* from the Recording toolbar. TestComplete will process the recorded test commands and save them to a test.

Until now, I have finished the keyword testing, Please return the TestComplete window. TestComplete shows the recorded keyword in the Keyword Test editor.

A noteworthy little detail is that we need to study how to refine the Recorded Test. You can run the recorded test directly, but we suggest to adjust the test to make it more stable. For instance, When you record

user actions, like mouse clicks, TestComplete records coordinates where the mouse click is performed. Coordinates that TestComplete records for one browser window size will become incorrect when you run your test for another browser window size. To simulate all click actions in the top left corner of the appropriate controls, So we will modify click operations (see Figure 2.18) in keyword editor. Firstly, find the operation is Click then click the ellipsis button in the Value column, it will open the Operation Parameters dialog. Secondly, In the dialog (see Figure 2.19), fill "1" as ClientX and ClientY's parameter separately, this means one click at the top left corner of the control. If you wanna fill "-1" as ClientX and ClientY's parameter separately, this means one click the center of the control. Finally, save your setting operation.

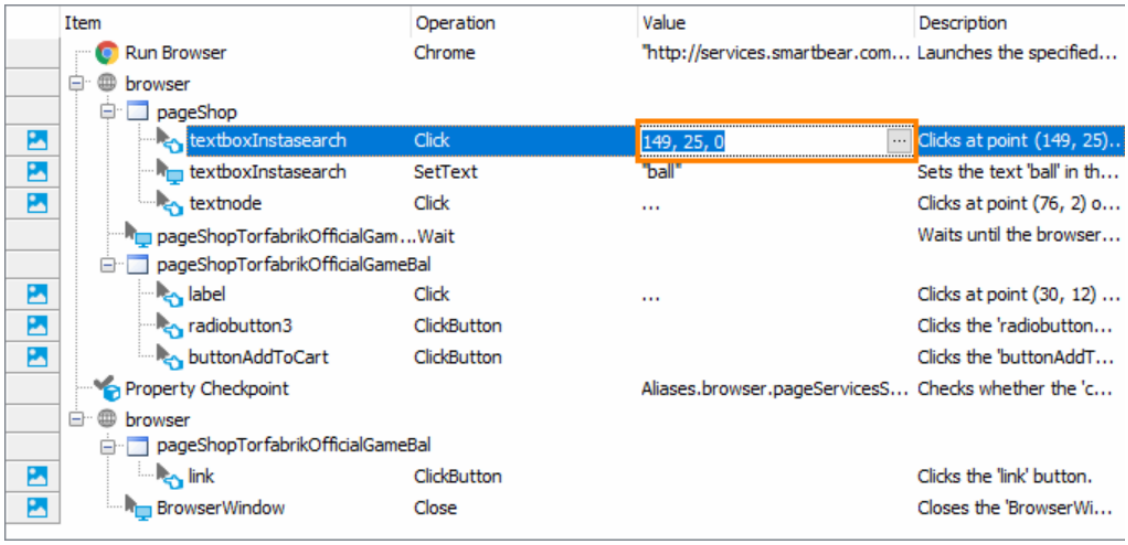


Figure 2.18. Adjust Action with TestComplete

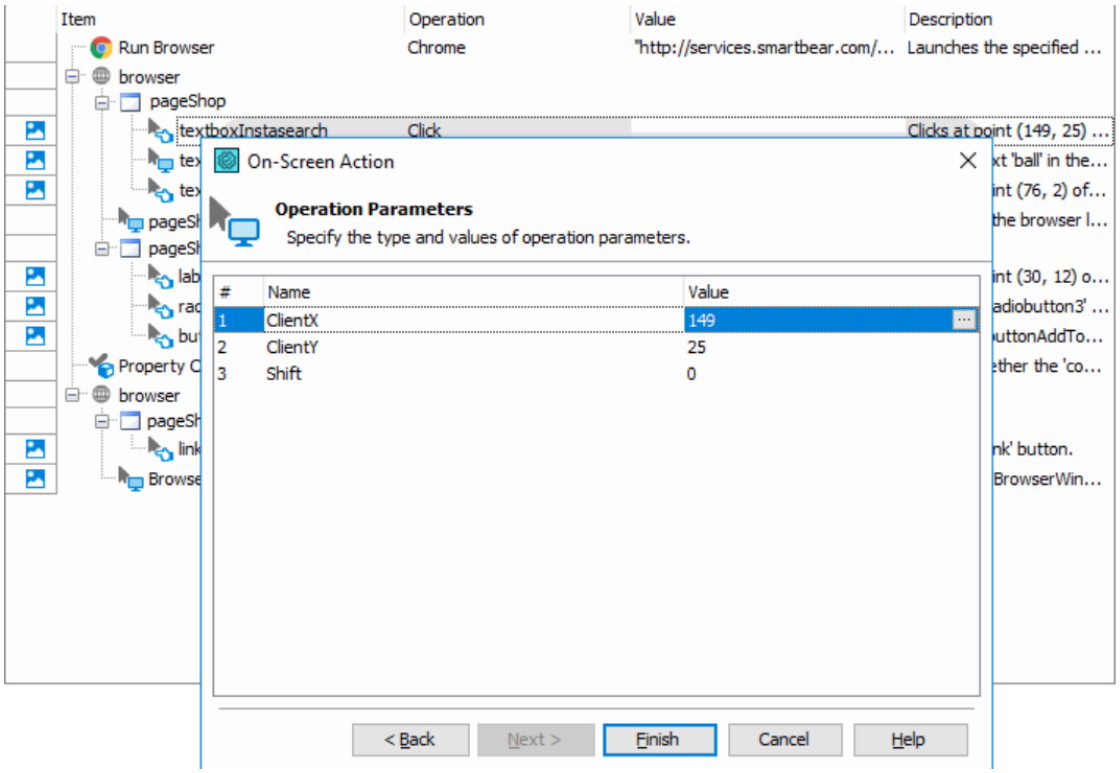
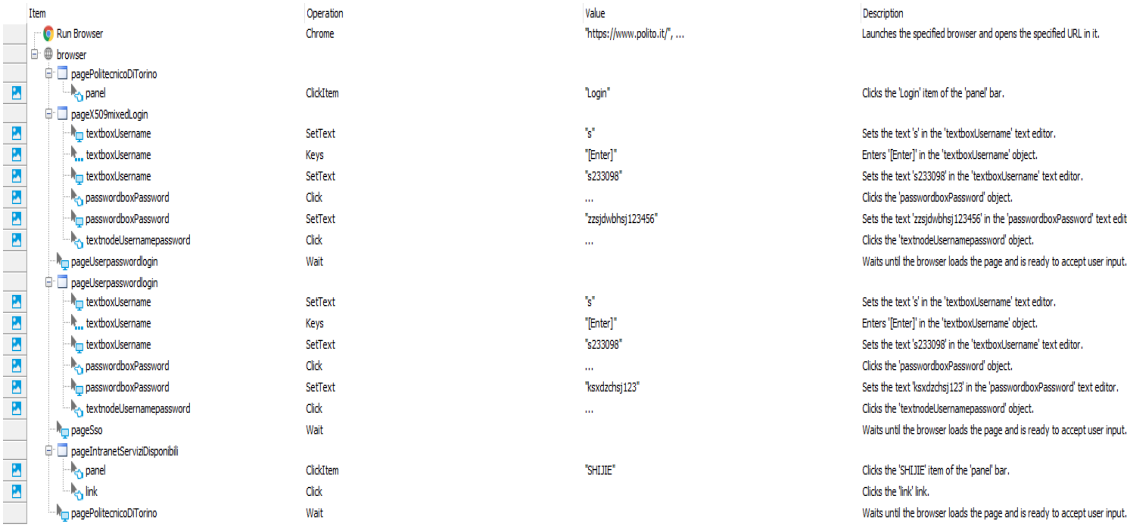


Figure 2.19. Adjust Action2 with TestComplete

2.6.3 Example test suite with TestComplete

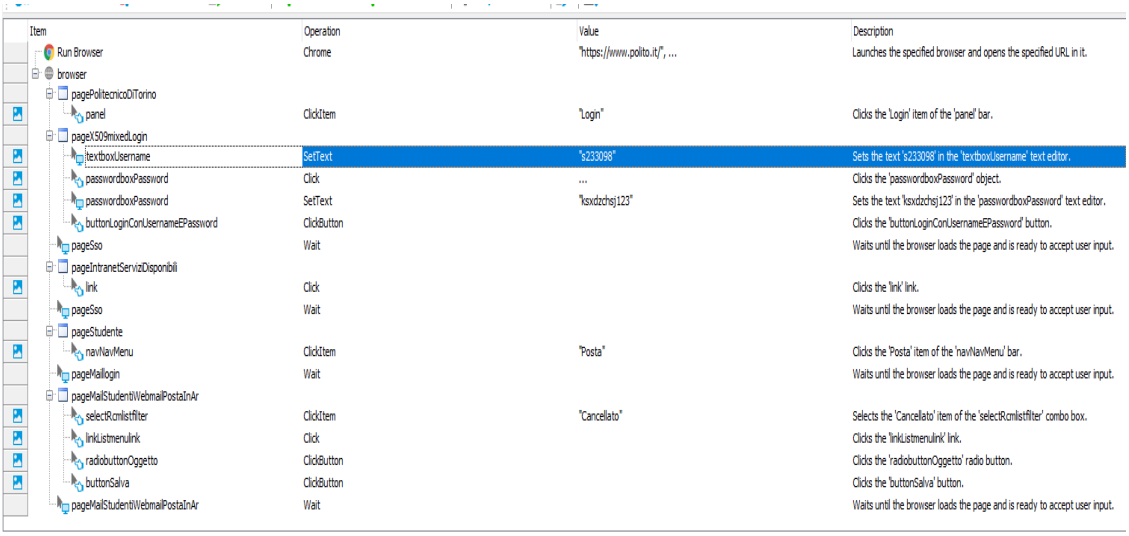
There are the test area Screenshots of five test cases applying the TestComplete testing tool.



Item	Operation	Value	Description
Run Browser	Chrome	"https://www.polito.it/", ...	Launches the specified browser and opens the specified URL in it.
browser			
pagePolitecnicoDiTorino			
panel	ClickItem	"Login"	Clicks the 'Login' item of the 'panel' bar.
pageX509mixedLogin			
textboxUsername	SetText	"s"	Sets the text 's' in the 'textboxUsername' text editor.
textboxUsername	Keys	"[Enter]"	Enters '[Enter]' in the 'textboxUsername' object.
textboxUsername	SetText	"s233098"	Sets the text 's233098' in the 'textboxUsername' text editor.
passwordboxPassword	Click	...	Clicks the 'passwordboxPassword' object.
passwordboxPassword	SetText	"zzsjdwbhsj123456"	Sets the text 'zzsjdwbhsj123456' in the 'passwordboxPassword' text edit
textnodeUsernamepassword	Click	...	Clicks the 'textnodeUsernamepassword' object.
pageUserpasswordlogin	Wait		Waits until the browser loads the page and is ready to accept user input.
pageUserpasswordlogin			
textboxUsername	SetText	"s"	Sets the text 's' in the 'textboxUsername' text editor.
textboxUsername	Keys	"[Enter]"	Enters '[Enter]' in the 'textboxUsername' object.
textboxUsername	SetText	"s233098"	Sets the text 's233098' in the 'textboxUsername' text editor.
passwordboxPassword	Click	...	Clicks the 'passwordboxPassword' object.
passwordboxPassword	SetText	"ksxdzchsj123"	Sets the text 'ksxdzchsj123' in the 'passwordboxPassword' text editor.
textnodeUsernamepassword	Click	...	Clicks the 'textnodeUsernamepassword' object.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageIntranetServiziDisponibili			
panel	ClickItem	"SHIJIE"	Clicks the 'SHIJIE' item of the 'panel' bar.
link	Click		Clicks the 'link' link.
pagePolitecnicoDiTorino	Wait		Waits until the browser loads the page and is ready to accept user input.

Figure 2.20. Login with TestComplete

Figure 2.20 shows the Polito web app login procedure. For a grouping node: "pageX509mixedLogin", Firstly I try to use a wrong password operation to verify the login page, so it did not load a new page successfully (It shows the window named "pageUserpasswordlogin"). Next, I set correct username and password operations, It shows the window named "pageSso".



Item	Operation	Value	Description
Run Browser	Chrome	"https://www.polito.it/", ...	Launches the specified browser and opens the specified URL in it.
browser			
pagePolitecnicoDiTorino			
panel	ClickItem	"Login"	Clicks the 'Login' item of the 'panel' bar.
pageX509mixedLogin			
textboxUsername	SetText	"s233098"	Sets the text 's233098' in the 'textboxUsername' text editor.
passwordboxPassword	Click	...	Clicks the 'passwordboxPassword' object.
passwordboxPassword	SetText	"ksxdzchsj123"	Sets the text 'ksxdzchsj123' in the 'passwordboxPassword' text editor.
buttonLoginConUsernameEPassword	ClickButton		Clicks the 'buttonLoginConUsernameEPassword' button.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageIntranetServiziDisponibili			
link	Click		Clicks the 'link' link.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageStudente			
navNavMenu	ClickItem	"Posta"	Clicks the 'Posta' item of the 'navNavMenu' bar.
pageMaillogin	Wait		Waits until the browser loads the page and is ready to accept user input.
pageMailStudentiWebmailPostaInAr			
selectRcmlistfilter	ClickItem	"Cancellato"	Selects the 'Cancellato' item of the 'selectRcmlistfilter' combo box.
linkListmenulink	Click		Clicks the 'linkListmenulink' link.
radiobuttonOggetto	ClickButton		Clicks the 'radiobuttonOggetto' radio button.
buttonSalva	ClickButton		Clicks the 'buttonSalva' button.
pageMailStudentiWebmailPostaInAr	Wait		Waits until the browser loads the page and is ready to accept user input.

Figure 2.21. Drop-down List with TestComplete

Figure 2.21 shows the procedure of checking drop-down lists in the mailbox. For a grouping node “pageMailStudentiWebmailPostaInAr”, We apply “ClickItem” on drop-down list object “selectRcmlistfilter” to select “Cancellato” option.

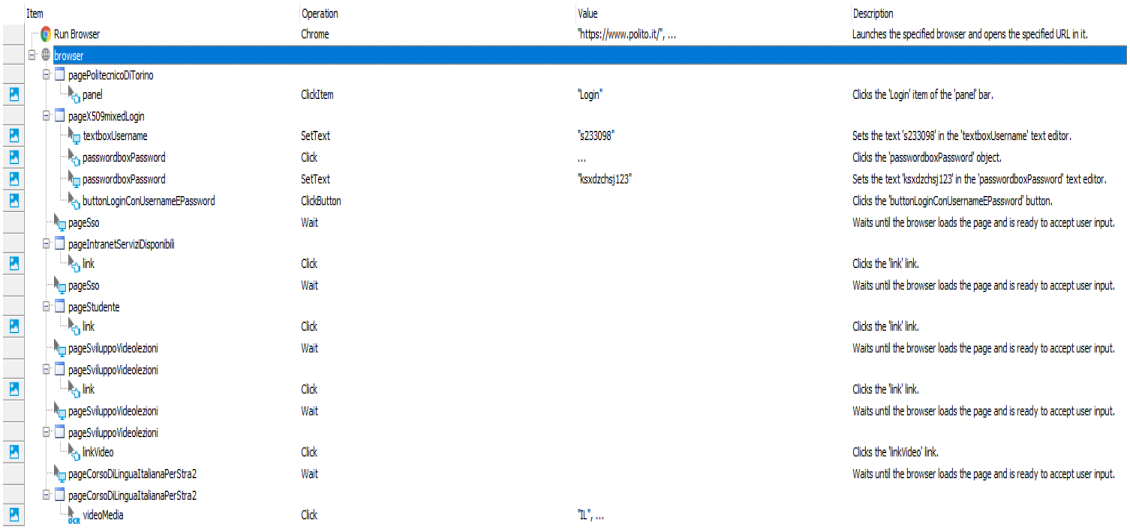
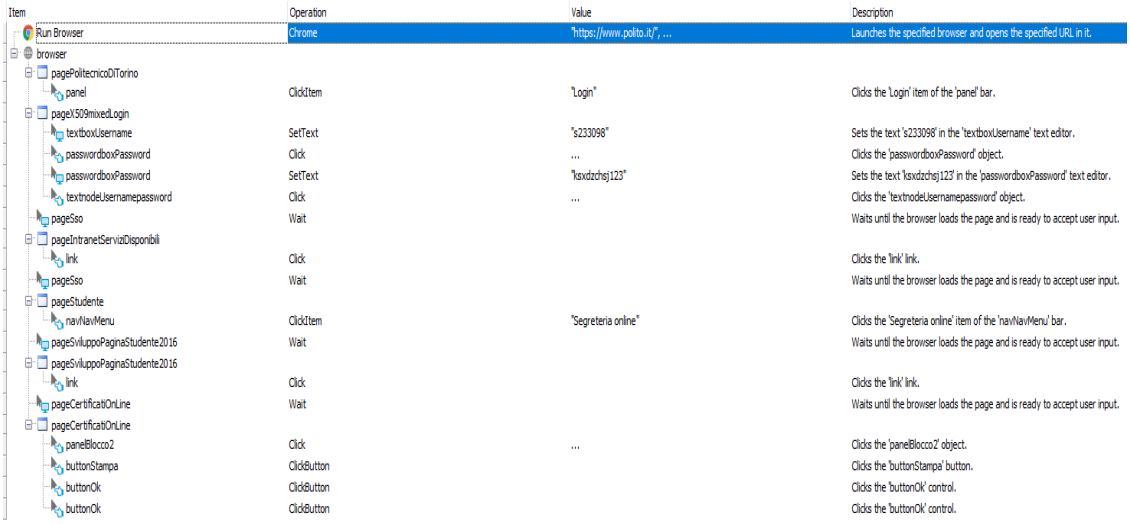


Figure 2.22. Operate Video with TestComplete

Figure 2.22 shows the procedure for opening a course video. For a grouping node: “pageSviluppoVideolezinoi”, I do click operation on “linkVideo” object. This action corresponds to the click “Video” operation.



Item	Operation	Value	Description
Run Browser	Chrome	"https://www.polito.it/", ...	Launches the specified browser and opens the specified URL in it.
browser			
pagePolitecnicoDiTorino			
panel	ClickItem	"Login"	Clicks the 'Login' item of the 'panel' bar.
pageX509mixedLogin			
textboxUsername	SetText	"s233098"	Sets the text 's233098' in the 'textboxUsername' text editor.
passwordboxPassword	Click	...	Clicks the 'passwordboxPassword' object.
passwordboxPassword	SetText	"ksxdzchsj123"	Sets the text 'ksxdzchsj123' in the 'passwordboxPassword' text editor.
textnodeUsernamepassword	Click	...	Clicks the 'textnodeUsernamepassword' object.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageIntranetServiziDisponibili			
link	Click		Clicks the 'link' link.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageStudente			
navNavMenu	ClickItem	"Segreteria online"	Clicks the 'Segreteria online' item of the 'navNavMenu' bar.
pageSviluppoPaginaStudente2016	Wait		Waits until the browser loads the page and is ready to accept user input.
pageSviluppoPaginaStudente2016			
link	Click		Clicks the 'link' link.
pageCertificatiOnLine	Wait		Waits until the browser loads the page and is ready to accept user input.
pageCertificatiOnLine			
panelBlocco2	Click	...	Clicks the 'panelBlocco2' object.
buttonStampa	ClickButton		Clicks the 'buttonStampa' button.
buttonOk	ClickButton		Clicks the 'buttonOk' control.
buttonOk	ClickButton		Clicks the 'buttonOk' control.

Figure 2.23. Print Certificate with TestComplete

Figure 2.23 shows the procedure for opening a course video. For a grouping node: "pageCertificatiOnline", I do click operation on object "panelBlocco2", then do ClickButton operation on object "buttonStampa". The actions correspond to select "Career" then click "Stampa" button.

Item	Operation	Value	Description
textboxUsername	Keys	"[Enter]"	Enters '[Enter]' in the 'textboxUsername' object.
textboxUsername	SetText	"s233098"	Sets the text 's233098' in the 'textboxUsername' text editor.
passwordboxPassword	Click	...	Clicks the 'passwordboxPassword' object.
passwordboxPassword	SetText	"ksxdzchsj123"	Sets the text 'ksxdzchsj123' in the 'passwordboxPassword' text editor.
buttonLoginConUsernameEPassword	ClickButton		Clicks the 'buttonLoginConUsernameEPassword' button.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageIntranetServiziDisponibili			
link2	Click		Clicks the 'link2' link.
pageSso	Wait		Waits until the browser loads the page and is ready to accept user input.
pageMailStudentiWebmailPostaInAr			
linkRcmbtn111	Click		Clicks the 'linkRcmbtn111' link.
pageMailStudentiWebmailPostaInAr	Wait		Waits until the browser loads the page and is ready to accept user input.
pageMailStudentiWebmailPostaInAr			
textareaDestinatario	Keys	"1372937384;[BS]@qq.com"	Enters '1372937384;[BS]@qq.com' in the 'textareaDestinatario' object.
linkCcLink	Click		Clicks the 'linkCcLink' link.
textareaCc	Click	...	Clicks the 'textareaCc' object.
textareaCc	Keys	"s233098@studt[BS]enti.polito.it"	Enters 's233098@studt[BS]enti.polito.it' in the 'textareaCc' object.
textboxOggetto	Click	...	Clicks the 'textboxOggetto' object.
textboxOggetto	SetText	"welcome to italy"	Sets the text 'welcome to italy' in the 'textboxOggetto' text editor.
textareaComposebody	Drag	60, 23, 82, 23	Drags the 'textareaComposebody' object.
textareaComposebody	Keys	"hi,everybody\"[BS]!!"	Enters 'hi,everybody"[BS]!!' in the 'textareaComposebody' object.
buttonAllegaUnFile	ClickButton		Clicks the 'buttonAllegaUnFile' button.
dlg_			
btn_	ClickButton		Clicks the 'btn_' button.
pageMailStudentiWebmailPostaInAr			
buttonAnnulla	ClickButton		Clicks the 'buttonAnnulla' button.
linkRcmbtn109	Click		Clicks the 'linkRcmbtn109' link.
pageMailStudentiWebmailPostaInAr	Wait		Waits until the browser loads the page and is ready to accept user input.

Figure 2.24. Write Email with TestComplete

Figure 2.24 shows the procedure for opening a course video. For a grouping node: "pageMailStudentiWebmailPostaInAr", firstly I enter

"1372937384@qq.com" using "Keys" operation as the content of the receiver object "textareaDestinatario". Then click on object "linkCcLink" for adding other receivers, so I also enter "s233098@studenti.polito.it" in object "textareaCc". Next, I write "welcome to Italy" in object "textboxOggetto" but apply "SetText" operation. Finally, I enter the content "Hi,everybody" in the object "textareaComposebody".

2.7 EyeAutomate

EyeStore includes many tools for visual GUI testing, such as EyeAutomate, EyeStudio, EyeServer, etc. I will focus on the EyeAutomate and EyeStudio tools for the papers. EyeAutomate is a visual script runner. Its characteristics describe as follows: This technology can work on all apps (desktop, web, and mobile) that have a graphical user interface. EyeAutomate fits in the system test phase and acceptance test phase. EyeAutomate does not depend on any third-party libraries, can be used on any system that supports Java from Oracle, and all commands in EyeAutomate are implemented in java. You can see commands are loaded and displayed in the command menu of EyeStudio workstation. EyeAutomate features the image recognition algorithm. It combines pixel- and vector-based image recognition with AI.

Pay attention: A license is required when EyeAutomate does not run from the Studio.

I will apply EyeStudio - a visual scripter editor, so let us move on to understand it. EyeStudio helps create and maintain your visual scripts. When editing a script, it could be manual, or semi-automatic, or automatic operation. When running the visual script directly from EyeStudio or standalone using the EyeAutomate script runner or the EyeServer script runner service.

2.7.1 SetUp

At the beginning, I follow this link to download EyeAutomate and EyeStudio (they are in one .jar named SetupEyeAutomate2.X.jar): https://eyeautomate.com/download-eyeautomate/.

I already have installed Java 8 and downloaded SetupEyeAutomate2.X.jar, so I directly install them by clicking their executable file.

2.7.2 How it works

Now I start to create the first test script when I launch EyeStudio.jar. I will create a test script about opening the Chrome browser and typing www.google.com and searching “Politecnico di Torino” in Google text search box.

First, we need to know:

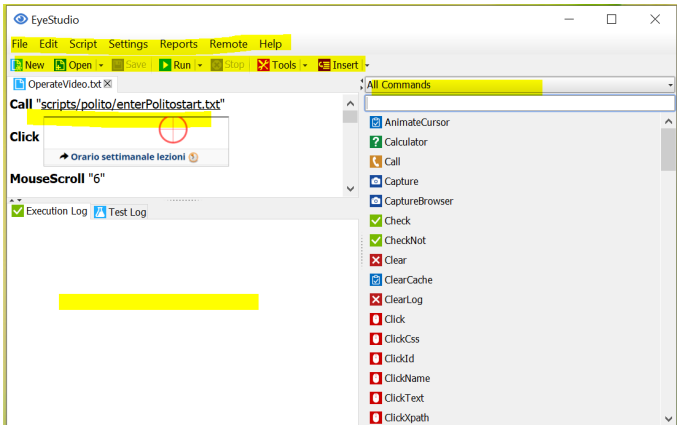


Figure 2.25. EyeStdio Workstation

EyeStdio window separates five components: (see Figure 2.25)

1. Menu bar
2. Toolbar
3. Test script editor pane
4. Execution log, Test log pane
5. Commands pane

I could create a new test case through *File* > *new* in the menu bar or *New* in the Toolbar, and named First.txt. Next, launch a chrome browser using the command: OpenBrowser “Chrome” (call OpenBrowser command from Commands pane). Then, call the website we wanna go: GetUrl “http://www.google.com/” (call GetUrl command from Commands pane). Once we find the name of a search text box after inspecting the website HTML code. Then I do click operation: ClickName “q” (call ClickName command from Commands panel), The Mouse cursor will focus on the search

text box, then try to type keywords: Type “Politecnico di Torino [ENTER]” (call Type command from Commands pane).

There are different ways to write this test, such that combine images, text commands, data, and widgets. Above is just one example of commands usage.

2.7.3 Example test suite with EyeAutomate

A script in EyeAutomate is visually readable and consists of commands, images, data, and widgets. A visual script may consist of four types of files:

- Scripts (.txt extension)
- Images (.png extension)
- Data (.csv extension)
- Widgets (.wid extension)

Figure 2.26. Login with EyeStdio

The login script (see Figure 2.26) mainly uses commands, images.

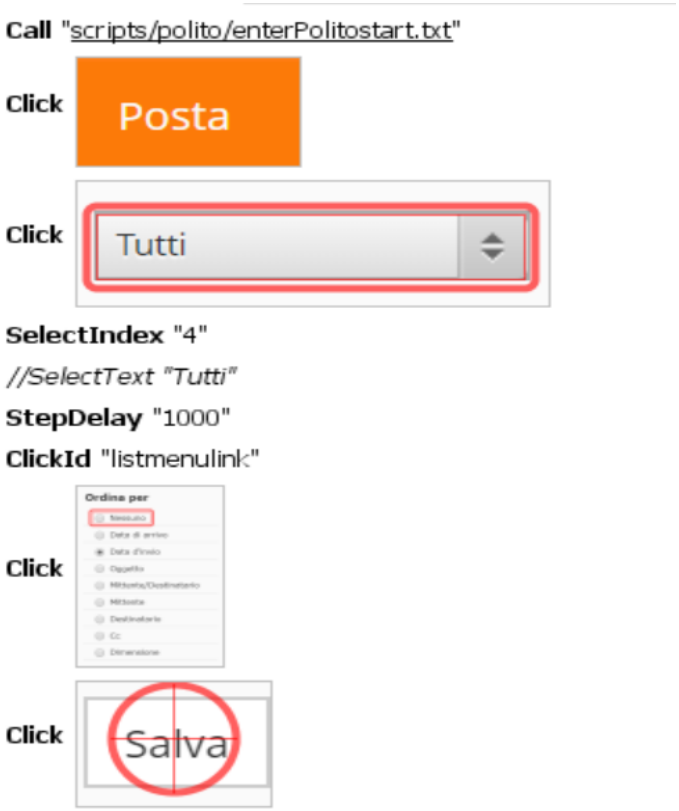


Figure 2.27. Drop-down List with EyeStdio

Before checking the drop-down list (see Figure 2.27), I have already logged in in the enterPolitostart.txt script (see Figure 2.28), which means that it first executes enterPolitostart.txt and then proceeds to the next command. It uses the SelectIndex command to perform the selection operation of the drop-down list.

Figure 2.28. Polito Starting with EyeStdio

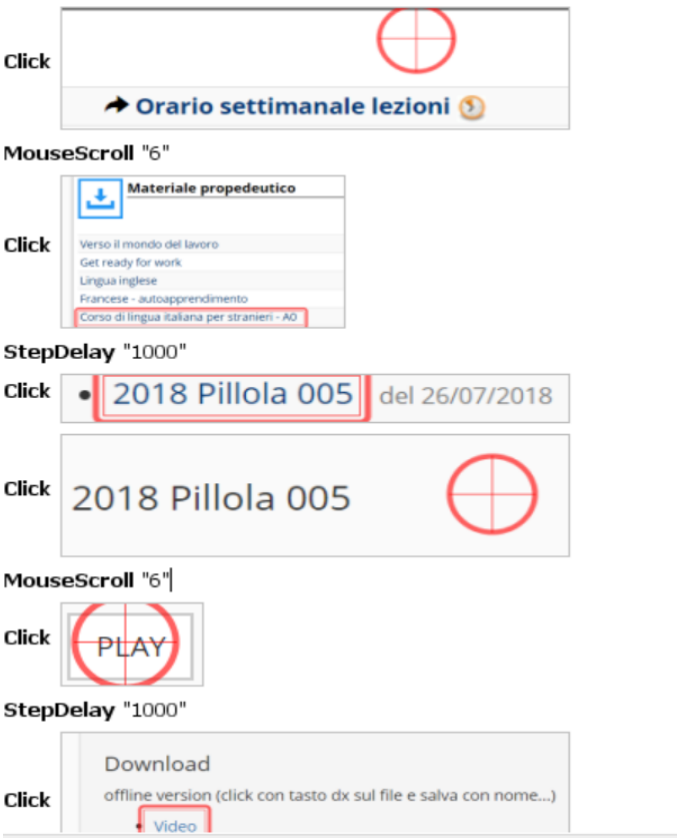


Figure 2.29. Operate Video with EyeStdio

This test firstly executes the enterPolitostart.txt script(see Figure 2.28). During opening a video (see Figure 2.29), here I use the MouseScroll command to simulate scrolling the screen operation.

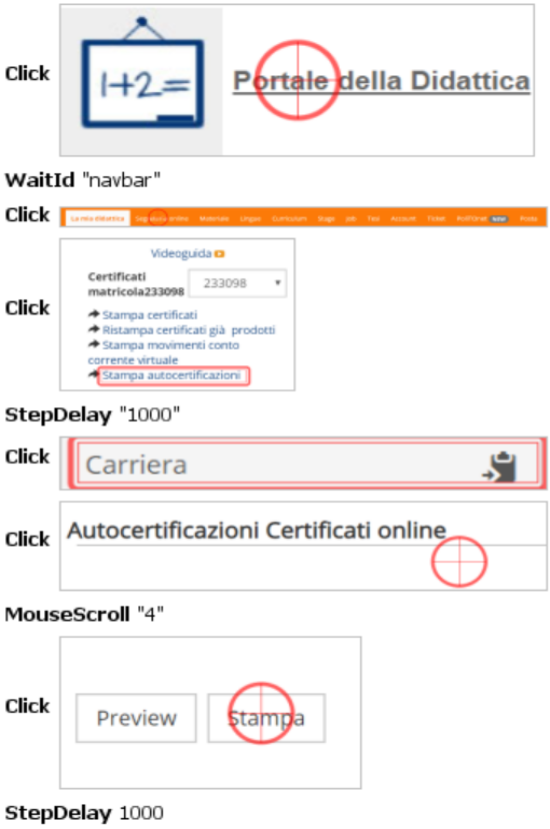


Figure 2.30. Print Certificate with EyeStdio

This test firstly executes the enterPolitostart.txt script (see Figure 2.28). The WaitId command here is used to wait until you see an element with id=navbar on the new page (see Figure 2.30).

Figure 2.31. Write Email1 with EyeStdio

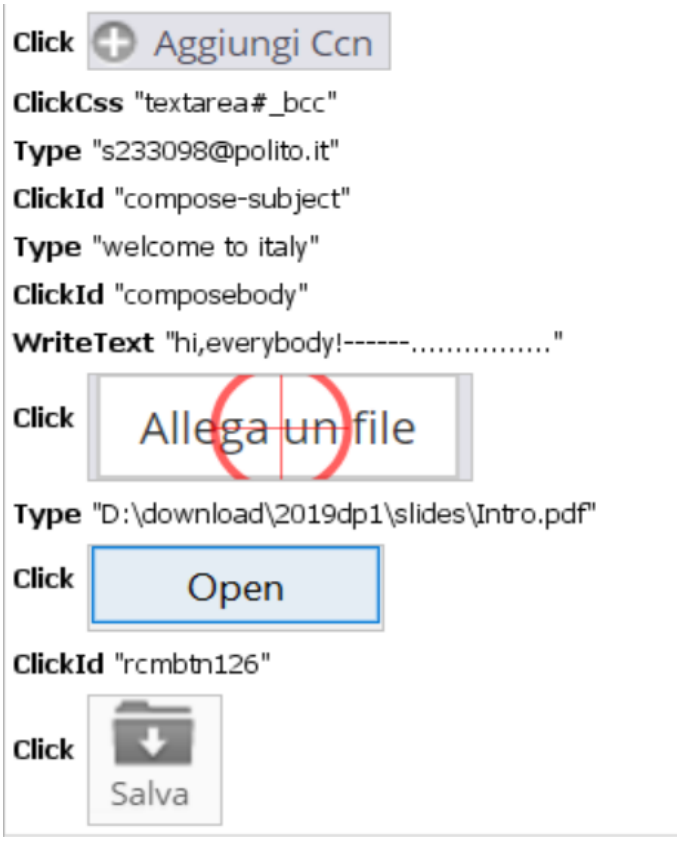


Figure 2.32. Write Email2 with EyeStdio

Here (see Figure 2.31, Figure 2.32) it combined commands and images in EyeStudio for writing an email.

Chapter 3

Empirical Evaluation

3.1 Fragility concept

In a simple, fragility means that quality becomes easily broken or damaged. We can see the meaning in the dictionary.

It is the direct responsibility of the software tester to sniff out fragility.

1. Software is "great" in size can suffer from fragility because one crisis can send the house of cards toppling down.

2. We can identify a fragile software by its inability to deal with stress.

   For software systems, the business is in a constantly changing environment and the software needs to adapt to the business needs quickly. Otherwise, it is obsolete. Apart from development time challenges, there are also runtime challenges for software systems too. Consequently, they are under stress by end-users and other systems. Then fragility could happen. Imagine a software project if it is not easy to extend, modify and deploy to the production environment. Or a system that is not able to handle unexpected user inputs or external system failures and breaks easily.

   That's a fragile system that is harmed by stress and penalized by change.

3. On the other hand, overoptimization also could be a reason for fragile software. The world of software testing is obsessed with automated checking, "green" results don't mean everything of software is fine, we can't ignore the fact that randomness is the rule, not the exception.

4. Changing would make software fragile when one line of code is changed, automation designed to find old forms of failure could be ineffective.

Test case robustness is one decisive factor of test quality. A test case is believed to be fragile when a small change in the application layout causes test cases to fail. To meet new requirements, add new functionalities, fix bugs, etc, developers usually apply changes to their web applications. Leotta et al, [17] defines two types of changes that can be applied over web applications: (1) Structural changes and (2) Logical Changes.

Structural changes refer to all the changes that modify the page layout and structure such as restyling or changing a web element's locator (There are multiple types of locators: ID, name, CSS selectors, XPath, DOM locators, etc.). Web element locators have emerged as the main cause of fragility of tests, confirming previous anecdotal findings, [18, 19]. Researchers have discussed and studied the fragility of web element locators, [20, 21, 22, 23]. Montoto et al. Logical changes refer to all the changes modifying the logic of the web application under test such as functionality addition, functionality modification, functionality deletion, etc, [17].

5. Besides, test cases created for one particular browser can easily break when executed on a different web browser.

Apart from this, Researchers adopt taxonomies to summary the cause of test case fragility. Researchers have presented various fault taxonomies. Bruning et al. [24] present a fault taxonomy for service-oriented architectures. Hayes [25] provides a fault taxonomy for NASA software requirements. Mariani [26] presents a taxonomy for component-based software systems. Chan et al. [27] provide a taxonomy for web service compositions. Hummer et al. [28] present a taxonomy for event-based systems. None of these taxonomies consider web applications.

A few papers present fault taxonomies for web applications. Ocariza et al. [29, 30] characterize the classes of error messages output by JavaScript in web applications and the classes of faults found in JavaScript. Ricca and Tonella [31] present a preliminary fault taxonomy for web applications, consisting of a single level of general fault categories. Marchetto et al. [32] present a taxonomy of web application faults. None of these papers consider test breakages.

To our knowledge, a paper that provides a detailed classification of how tests of web applications may break is by Mouna Hammonudi [33] and Figure 3.1 presents a graphical view of his taxonomy of causes of test breakages.

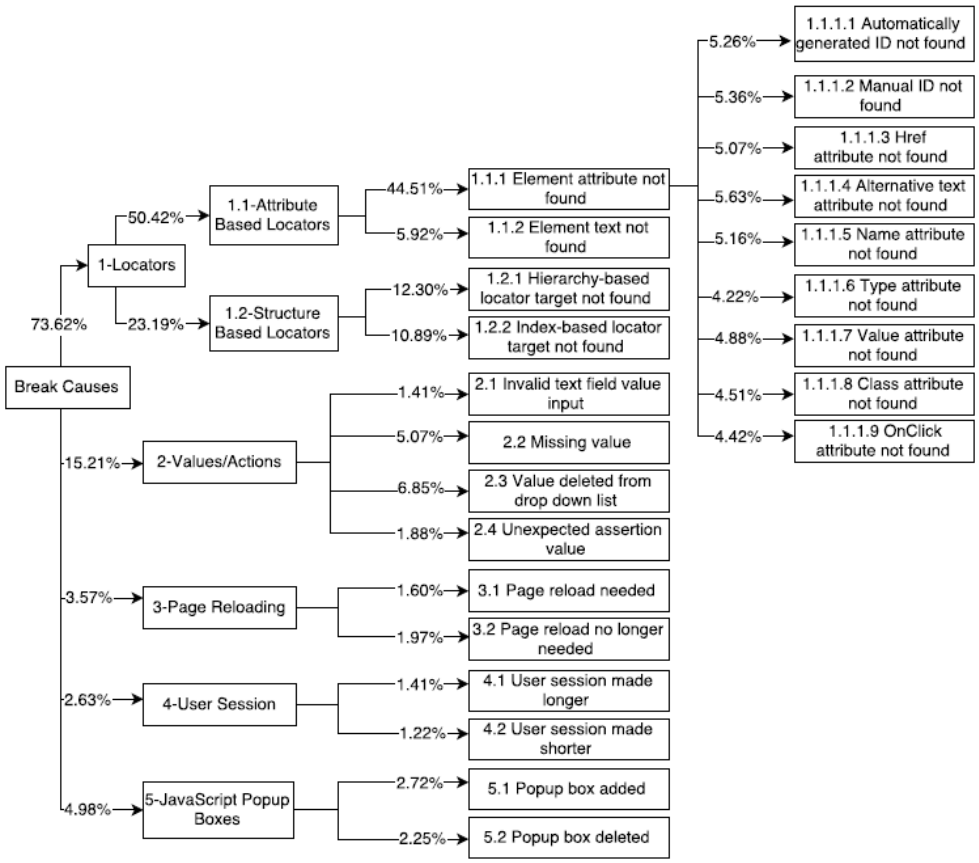


Figure 3.1. Taxonomy

Figure 3.1 points out the major test breakage causes are related to locators (50.42% is attribute-based locators and 23.19% is structure-based locators). The author gives the taxonomy which focuses on the proximal causes of test breakages. His taxonomy classifies the causes of test breakage that most nearly impact the test code and does not attempt to categorize the types of changes associated with the evolution of the web application code.

We can also talk about fragility from another side, which is tool fragility. The different types of tools to test web applications can be applied. First-generation testing tools rely on on-screen coordinates to locate and manipulate web elements of the web application. Screen resolution and window size will be related as the major reason for fragile test suites. Second-generation testing tools offer simple web element selection mechanisms to counterbalance the drawbacks of first-generation ones. The web element of interest is

located according to its position within the DOM tree of the web page under the test, [34]. Tests that are created using second-generation tools are fragile as well, given that their execution is dependent on the DOM tree of the web page under test, [34]. Simple changes in the DOM tree of a web application could cause fragile testing. Third-generation testing tools base on image recognition of web elements within the user interface of the web application. Tests created using third-generation tools are fragile as well, because simple changes (the size and the color of the web element) to visual appearance could cause fragile testing.

As mentioned above, fragility is inevitable in the world. By studying fragility, we can better serve test repair and better develop testing tools and it can help engineers avoid the need for test repairs by alerting them to changes that may break tests, etc.

3.2 Object of study: Dolibarr

Dolibarr ERP&CRM uses a software package to manage your businesses such as CRM/Sales, Human Relationship, CMS/Website/E-Commerce, Product and Stock, Finance and Billing, Marketing, Productivity, Integration/Development (see Figure 3.2). Whatever is your business management needs (sales, human resources, logistic, stock, invoicing, accounting, manufacturing, e-mailings, etc), you can set up the application to match your needs, and only one thing you need to do is to enable the feature you expect.

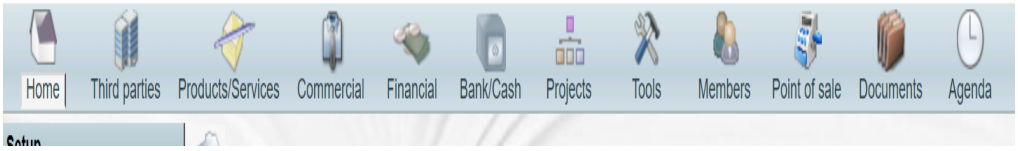


Figure 3.2. Dolibarr Navigation

Dolibarr ERP&CRM provides default modules (business or technical modules) with standard distribution (see Figure 3.2). You also can implement enhancements or personalization by adding third parties external modules. You can find them on the official market place of external addons (DoliStore), they may be free or charged.

Dolibarr ERP&CRM is an open-source and free model, serves as my experimental object for selenium WebDriver automated testing using java in several versions.

Now the number of releases/commits of Dolibarr is 102 until version 10.0.2. But I did have to exclude some. I only retain the last minor for each major release. The number of selected versions list as below:

- 2.9.0
- 3.9.4
- 4.0.6
- 5.0.7
- 6.0.8
- 7.0.5
- 8.0.6
- 9.0.4
- 10.0.2

In my experiment, I use a systematic and iterative procedure for Dolibarr. First, I install its initial version, V2.9.0, and familiarize ourselves with all of the functionalities provided by the application. Next, a test suite T is created for version V2.9.0. Then I execute that suite on the next version V3.9.4 and note each instance in which a test broke. The next step is followed by manually repairing each of the tests that were repairable. This is an iterative process because repairing one test breakage might allow that test to proceed further and break again later. The foregoing process was repeated for each selected version of Dolibarr until each had been tested and the causes of all test breakages had been noted. While I follow this process, try to repair the tests for a current version V, No attempt is made to check the next version until the repair process is complete.

3.3 Experiment

I have endeavored to implement the testing procedure on Dolibarr and carried out experiments. The implementation is on windows 10. The browser is Chrome 76 and uses corresponding ChromeDriver 76.0.3809.126.

Since the application requires an authentication phase, tests are intended to run sequentially, so just one authentication has to be performed.

There is an order relationship between some test cases: T7 had to be executed after T14; T11 had to be executed after T12, T13. T14 will create a new project and this new project is necessary as a test element in T7. T12 will create a financial account and T13 will create a bank account number, they all are necessary as test elements in T11.

For each selected version of the application, 19 tests are implemented, thus implementing a total of 171 tests for Dolibarr.

Each test case has been implemented in an automated test script using Selenium WebDriver and EyeAutomate. Among them, in particular, there are some special situations here, I need to explain here. When T7 of V2.9.0 has been checked the validity on V3.9.4, I found that all test elements in T7 are totally different and not applicable anymore for the next version V3.9.4 but keep the same functionality. So I decide to fix all elements of the test script for V3.9.4 but with the same scenario. Another thing is that T12 and T13 in V2.9.0 are processed in their respective web pages. But in the next version, they are integrated on the same page. But I can add a "modify" element to T13 to achieve the same effect.

3.4 Test case definition

Nineteen test cases are created to test Dolibarr. Table 3.1 shows the sample of test cases with their description.

Table 3.1. Description of Dolibarr Usage Scenarios

Test Case ID	Description
T1	User cannot login due to invalid credentials
T2	Fill a company/foundation info
T3	Create a new third party in third party module
T4	Create a new product in products/services module
T5	Create a new service in products/services module
T6	Create a new warehouse in products/services module
T7	Create a new contact in commercial module
T8	Create a new customer's order in commercial module
T9	Create a new intervention in commercial module
T10	Create a new donation in financial module
T11	Create a new trips and expenses in financial module
T12	Create a new financial account in bank/cash module
T13	Create a new bank account number in bank/cash module
T14	Create a new project in projects module
T15	Create a new task in projects module
T16	Set up member types in members module
T17	Create a new member in members module
T18	Log in authentication in point of sale module
T19	Create a new action/task in agenda module

Chapter 4

Results

4.1 Experiment results

In this part, I will discuss how the tests have been written for the individual versions of the application, and the defects and fragilities I have found for each one of them (see Figure 4.1). Besides, I also show the testing result by using the EyeAutomate automated testing tool (see Figure 4.16). These two graphs do not include release 2.9.0, for which the test cases were first defined and hence were all executable. The remainder of the part, I will highlight which test cases applied to subsequent versions and, on the converse, what changes have had to be made to adapt them.

To better describe these modifications between different versions, after referencing the literature, [33], I summarized the experimental results to a certain extent, as described below for Selenium and EyeAutomate separately. Throughout this part, let V be a version of Dolibarr, let t be a test created and executed on V, and let V' be a new version of V.

4.1.1 Result Under Selenium WebDriver

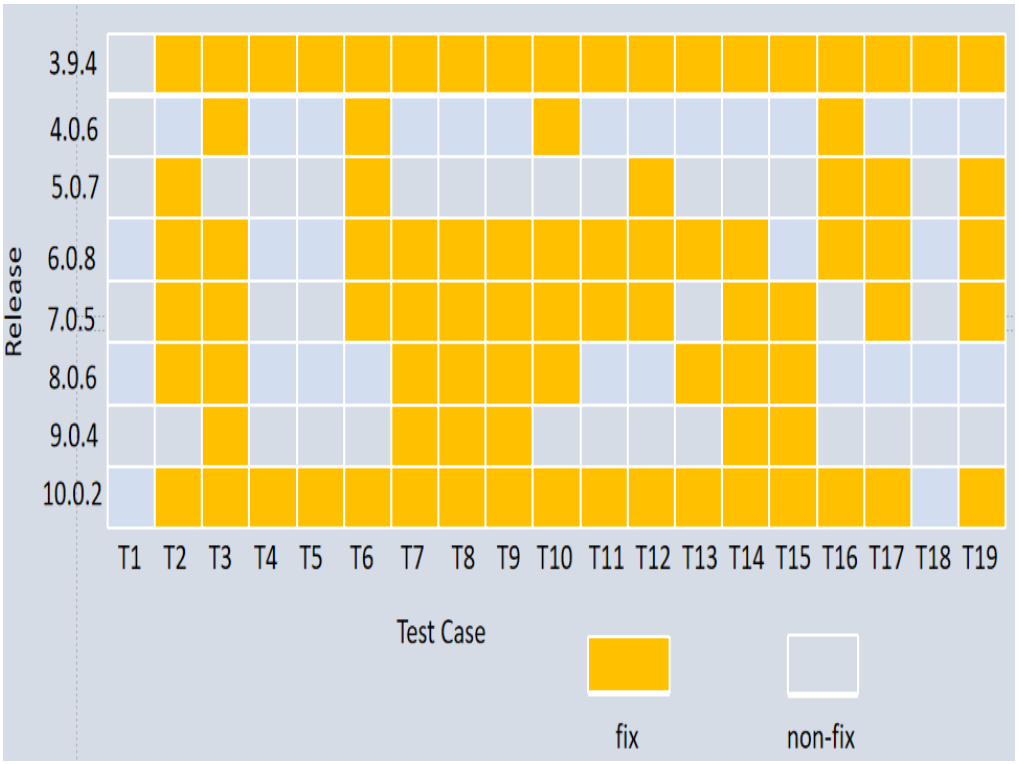


Figure 4.1. Modifications needed on various versions of Dolibarr, with Selenium

4.1.2 Summary of Dolibarr Change Types under Selenium

Based on the results of this experiment, some classifications were made in conjunction with Selenium.

- Name Attribute->Name/ID Attribute

  The modification would arise in a situation When t attempts to locate a web element e via a name attribute value that functioned (led to e) in V, but no longer functions in V'. For example, When T2 of V2.9.0 (see Figure 4.2) locates "Zip" "Town" text field in the application, they are altered in V3.9.4 such that name="cp" is now name="zipcode", name="ville" is now name="town" in HTML code shown in Figure 4.3. When T2 of v2.9.0 locates "Country" drop-down list, because of the non-existence of the name attribute in HTML code, it is altered in v3.9.4 such

that name=“pays_id” is now id=“s2id_selectcountry_id” in java code (Figure 4.4).

```
▼<tr class="impair">
    <td>Zip</td>
  ▼<td>
      <input name="cp" value="10142" size="10">
    </td>
  </tr>
▼<tr class="pair">
    <td>Town</td>
  ▼<td>
      <input name="ville" size="30" value="Milan">
    </td>
```

Figure 4.2. HTML code for T2 in V2.9.0

```
▼<tr class="impair">
  ▼<td>
      <label for="zipcode">Zip</label>
    </td>
  ▼<td>
      <input name="zipcode" id="zipcode" value="10142" size="10"> == $0
    </td>
  </tr>
▼<tr class="pair">
  ▼<td>
      <label for="town">Town</label>
    </td>
  ▼<td>
      <input name="town" id="town" size="30" value="Milan">
    </td>
  </tr>
```

Figure 4.3. HTML code for T2 in V3.9.4

```
// 18 | click | name=pays_id | |
Select country= new Select(driver.findElement(By.name("pays_id")));
country.selectByValue("3");
```

```
//country
WebElement element1 = driver.findElement(By.id("s2id_selectcountry_id"));
element1.click();
WebElement e2= driver.findElement(By.id("select2-results-1"));
e2.findElement(By.xpath("//*[@id=\"select2-results-1\"]/li[108]")).click();
```

Figure 4.4. Java code for T2 between V2.9.0 and V3.9.4

- Element in DOM Tree Change

The modification is caused when t attempts to locate an element e via a hierarchy-based locator l that functioned (led to e) in V, but no longer functions in V'. The causes of the modification mostly an ancestor of e in the DOM tree occur addition, deletion, modification, even totally change. For example, When T6 of V2.9.0 (see Figure 4.5) applies Select class which provides select and deselects options to test item "Country" (but above can't work in V3.9.4), so substitutes it with XPath locator in the current release (see Figure 4.6). Besides, for the Comment element, because the DOM tree structure changed (see Figure 4.7), usage of name locator has substituted by the usage of frames switching.

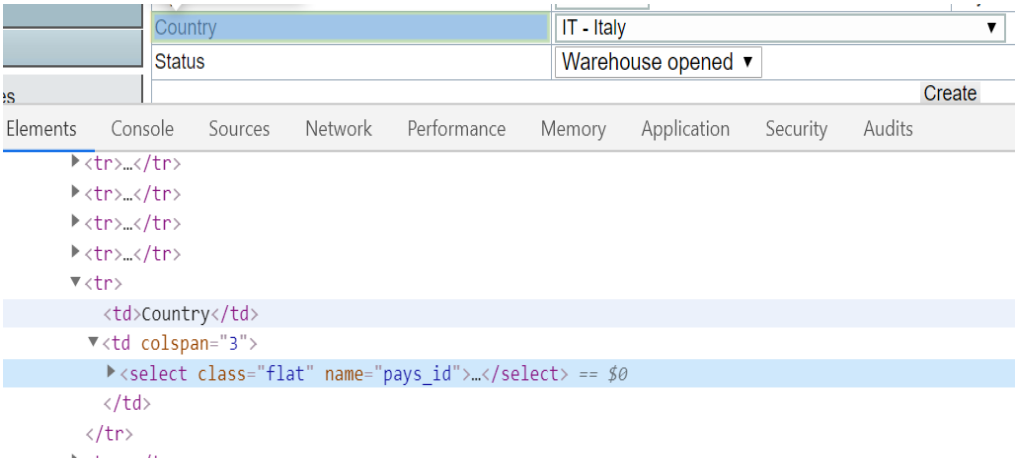


Figure 4.5. HTML code for T6 in V2.9.0

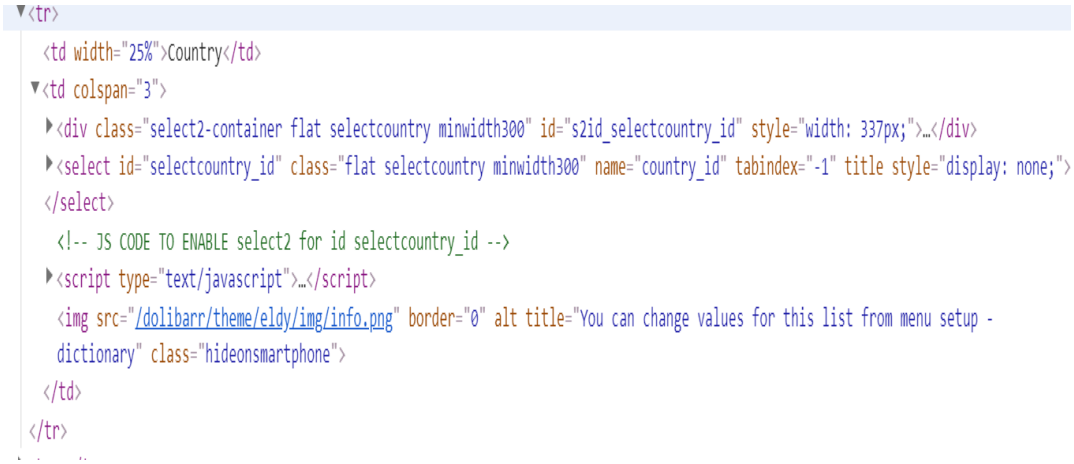


Figure 4.6. HTML code for T6 in V3.9.4

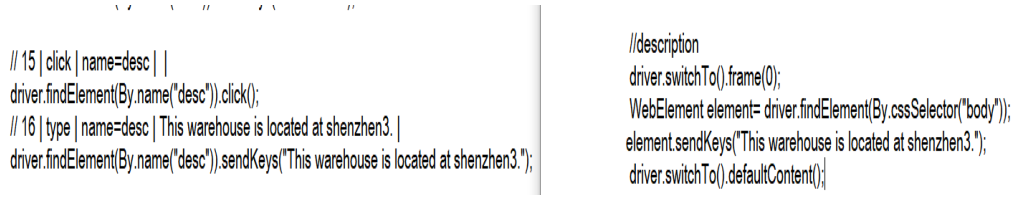


Figure 4.7. Java code for T6 between V2.9.0 and V3.9.4

- Link Text Change

  The modification would arise in a situation When t attempts to locate a link l via a *href* attribute value that functioned (led to l) in V, but no longer functions in V'. It could link text change, link deletion, addition, etc. For example, when T14 locates a link via the content "Valid" in V2.9.0 but is altered to "Validate" in V3.9.4 (see Figure 4.8). T8 of V2.9.0 attempts to locate a link via the content "Customer's order", but the link is deleted in V3.9.4 (see Figure 4.9).

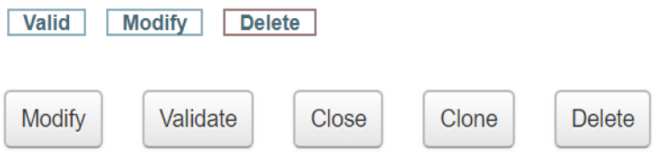


Figure 4.8. Capture Dolibarr for T14 between V2.9.0 and V3.9.4

```
driver.findElement(By.linkText("Customer\'s orders")).click();
driver.findElement(By.linkText("New order")).click();
driver.findElement(By.linkText("Customer")).click();
driver.findElement(By.linkText("Add order")).click();
driver.findElement(By.name("ref_client")).click();
driver.findElement(By.name("ref_client")).sendKeys("000005");
```

```
driver.findElement(By.linkText("New order")).click();
driver.findElement(By.name("ref_client")).click();
driver.findElement(By.name("ref_client")).sendKeys("000005");
```

Figure 4.9. Java code for T8 between V2.9.0 and V3.9.4

- Element Addition

  When I try to submit a webpage, the submission fails because some elements in the webpage are required. This means that you must add these elements to the current version. For example, the "Supplier" drop-down list in T3 of V2.9.0 may need to select one option "No" for adapting to V3.9.4, this item is necessary to add in V3.9.4 (see Figure 4.10) otherwise this web page can't be submitted.

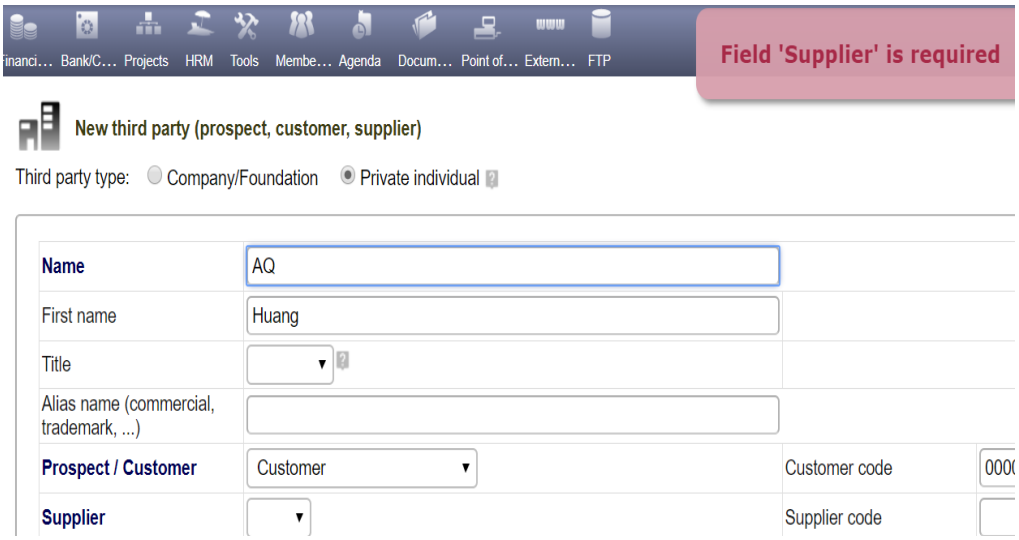


Figure 4.10. Capture Dolibarr for T3 between V2.9.0 and V3.9.4

- Element Deletion

  If there is no web page element e in V', then the test t of V will fail in V'. Therefore, I need to remove this element in version V'. For example, in T19, We can locate one web element "Type" via name attribute "actioncode" in V2.9.0 (see Figure 4.11), but this web element is deleted

in V3.9.4 (see Figure 4.12). Locating a web element “Action affected to” via name attribute “affectedto” in V2.9.0 (see Figure 4.11), this web element also is deleted in V3.9.4 (see Figure 4.12).

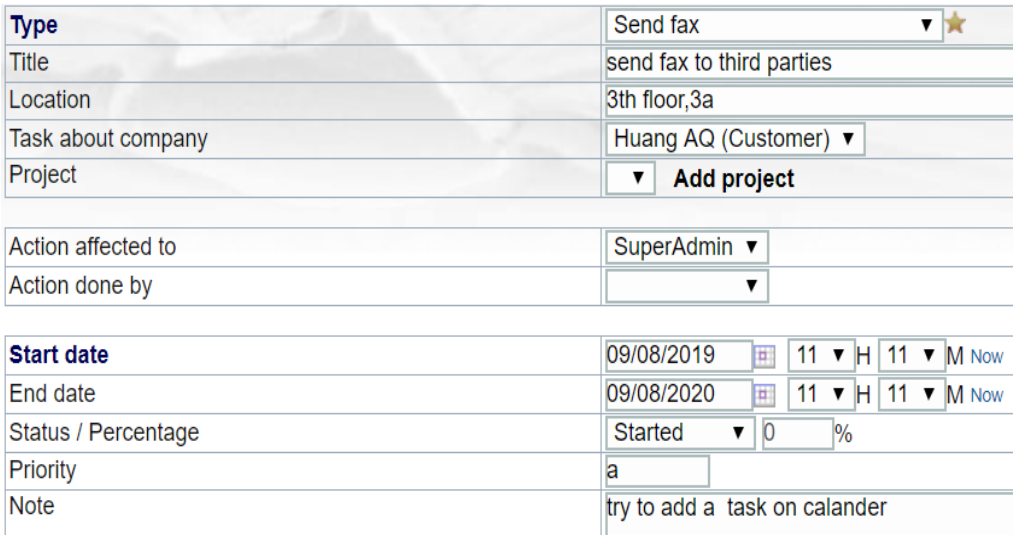


Figure 4.11. Capture Dolibarr for T19 in V2.9.0

- Element Text Change

  The modification is caused when t attempts to locate an element e via its text/value, using a text string that functioned correctly (led to e) in V, but no longer functions in V’. For example, when T12 of V2.9.0 (see Figure 4.13) locates the "Status" element, in the drop-down list option, the option text was changed from “Opened” to “Open” to fit the subsequent new version.

- ID Attribute->ID Attribute

  The modification would arise in a situation When t attempts to locate a web element e via an ID attribute value that functioned (led to e) in V, but no longer functions in V’. For example (see Figure 4.14), when T14 of V2.9.0 locates the “Date start” text field in the application, it is altered such that id=“project” is id=“projectstart” in V3.9.4.

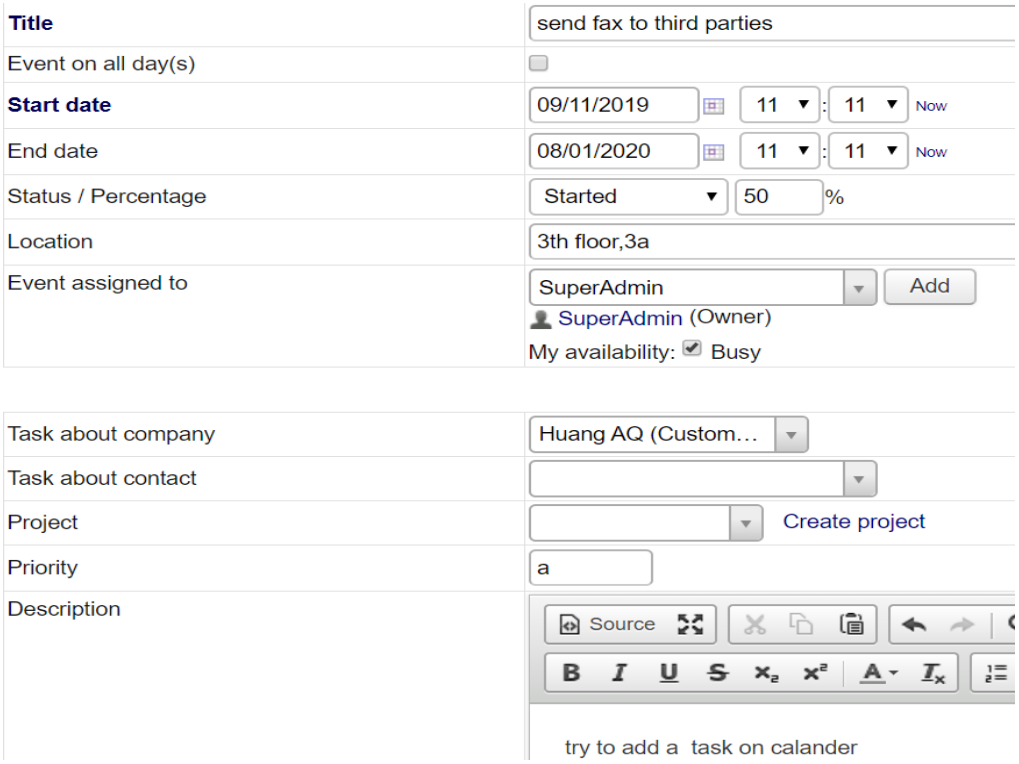


Figure 4.12. Capture Dolibarr for T19 in V3.9.4

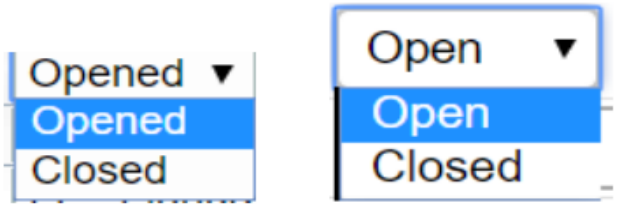


Figure 4.13. Capture Dolibarr for T12 between V2.9.0 and V3.9.4

```
// 18 | click | id=project | |
driver.findElement(By.id("project")).click();
// 19 | type | id=project | 29/09/2019 |
driver.findElement(By.id("project")).sendKeys("29/09/2019");
```

```
// 18 | click | id=project | |
driver.findElement(By.id("projectstart")).click();
// 19 | type | id=project | 29/09/2019 ||
driver.findElement(By.id("projectstart")).sendKeys("29/09/2019");
```

Figure 4.14. Java code for T14 between V2.9.0 and V3.9.4

- Indices of Element Change

  The modification is caused when t attempts to locate an element e via index-based locator l that functioned (led to e) in V, but no longer functions in V'. This could occur when elements add or delete to/from the web page that satisfies the same locator strategy of an element in V, but cause a change in the index of the element that t was intended to select. For example, for T2 of V5.0.7 (see Figure 4.15), from "Country" drop-down list to select "Italy(IT)" option, because the addition of elements to the web page that satisfies the same locator strategy of an element in V4.0.6, but causes a change in the index of the element that T2 was intended to select. So it is substituted from "//*[@id="select2-results-1"]/li[108]" to "//*[@id="select2-results-1"]/li[109]".

```
//country
WebElement element1 = driver.findElement(By.id("s2id_selectcountry_id"));
element1.click();
WebElement e2= driver.findElement(By.id("select2-results-1"));
e2.findElement(By.xpath("//*[@id=\"select2-results-1\"]/li[108]")).click();
```

```
//country
WebElement element1 = driver.findElement(By.id("s2id_selectcountry_id"));
element1.click();
WebElement e2= driver.findElement(By.id("select2-results-1"));
e2.findElement(By.xpath("//*[@id=\"select2-results-1\"]/li[109]")).click();
```

Figure 4.15. Java code for T2 between V4.0.6 and V5.0.7

4.1.3 Report on Dolibarr Test Cases under Selenium

- Dolibarr V3.9.4

  All of the test cases of V2.9.0 needs to fix on this version of the app, except T1: the authentication test.

Table 4.1. Report on V3 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T2,T3,T4,T5,T6,T8,T9,T10,T11,T12,T13,T14,T17,T19
Element in DOM Tree Change	TT2,T3,T4,T5,T6,T8,T9,T10,T11,T12,T14,T16,T17,T19
Link Text Change	T8,T11,T13,T14,T15,
Element Addition	T3,T11,T18
Element Deletion	T3,T4,T5,T6,T8,T10,T11,T19
Element Text Change	T4,T5,T6,T12
ID Attribute->ID Attribute	T14,T17
Indices of Element Change	NULL

- Dolibarr V4.0.6

Four tests out of nineteen need to fix in this version of the app. All tests written for V3.9.4 are compatible with this version, except T6: create a new warehouse in products/services module and T10: create a new donation in financial module and T16: set up member types in members module.

Table 4.2. Report on V4 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T3
Element in DOM Tree Change	NULL
Link Text Change	T10,T16
Element Addition	NULL
Element Deletion	NULL
Element Text Change	T6
ID Attribute->ID Attribute	NULL
Indices of Element Change	NULL

- Dolibarr V5.0.7

  Six tests out of nineteen need to fix in this version of the app. All tests written for V4.0.6 are compatible with this version, except T2: fill a company/foundation info And T6: create a new warehouse in products/services module and T12: create a new financial account in bank/cash module and T16: set up member types in members module and T17: create a new member in members module and T19: create a new action/task in agenda module.

Table 4.3. Report on V5 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T16
Element in DOM Tree Change	NULL
Link Text Change	NULL
Element Addition	NULL
Element Deletion	NULL
Element Text Change	T6,T19
ID Attribute->ID Attribute	T6
Indices of Element Change	T2,T6,T12,T17

- Dolibarr V6.0.8

Fourteen tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T4: create a new product in products/services module and T5: create a new service in products/services module and T15: create a new task in projects module and T18: login authentication in point of sale module are completely compatible with the ones written for release V5.0.7.

Table 4.4. Report on V6 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T12,T16
Element in DOM Tree Change	T12
Link Text Change	T2
Element Addition	NULL
Element Deletion	NULL
Element Text Change	T6
ID Attribute->ID Attribute	T2,T3,T6,T7,T8,T9,T10,T11,T12,T14,T17,T19
Indices of Element Change	NULL

- Dolibarr V7.0.5

  Thirteen tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T4: create a new product in products/services module and T5: create a new service in products/services module and T13: create a new bank account number in bank/cash module and T16: set up member types in members module and T18: login authentication in point of sale module is completely compatible with the ones written for release V6.0.8.

Table 4.5. Report on V7 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T2,T15
Element in DOM Tree Change	T2,T3,T6,T7,T8,T9,T10,T11,T12,T14,T15,T17,T19
Link Text Change	T2
Element Addition	NULL
Element Deletion	NULL
Element Text Change	T11
ID Attribute->ID Attribute	T2,T3,T6,T7,T7,T8,T9,T10,T11,T12,T14,T17
Indices of Element Change	NULL

- Dolibarr V8.0.6

Nine tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T4: create a new product in products/services module and T5: create a new service in products/services module and T6: create a new warehouse in products/services module and T11: create a new trips and expenses in financial module and T12: create a new financial account in bank/cash module and T16: set up member types in members module and T17: create a new member in members module and T18: login authentication in point of sale module and T19: create a new action/task in agenda module are completely compatible with the ones written for release V7.0.5.

Table 4.6. Report on V8 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T2
Element in DOM Tree Change	NULL
Link Text Change	T2,T3,T10,T13,T14
Element Addition	NULL
Element Deletion	T15
Element Text Change	T7,T8,T9,T15
ID Attribute->ID Attribute	T3
Indices of Element Change	NULL

- Dolibarr V9.0.4

Six tests out of nineteen need to fix in this version of the app. All tests written for V8.0.6 are compatible with this version, except T3: create a new third party in third party module and T7: create a new contact in commercial module and T8: create a new customer's order in commercial module and T9: create a new intervention in commercial module and T14: create a new project in projects module and T15: create a new task in projects module.

Table 4.7. Report on V9 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	T3
Element in DOM Tree Change	NULL
Link Text Change	T14
Element Addition	NULL
Element Deletion	NULL
Element Text Change	T7,T8,T9,T15
ID Attribute->ID Attribute	NULL
Indices of Element Change	NULL

- Dolibarr V10.0.2

  Seventeen tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T18: login authentication in point of sale module are completely compatible with the ones written for release V9.0.4.

  For T2, T3, T4, T5, T6, T7, T9, T10, T11, T13, T14, T16, T17, T19 occur scroll screen situations. For T2, T3, T4, T5, T8, T10, T12, T13, T14, T15, T16, T17, T19 click the remove button in the footer of the web page to enable test case running successfully.

Table 4.8. Report on V10 of Dolibarr, with Selenium

Situation	Occurred Test Case
Name Attribute->Name/ID Attribute	NULL
Element in DOM Tree Change	T5,T14
Link Text Change	NULL
Element Addition	NULL
Element Deletion	T3
Element Text Change	T7,T9,T15
ID Attribute->ID Attribute	NULL
Indices of Element Change	NULL

4.1.4 Result Under EyeAutomate

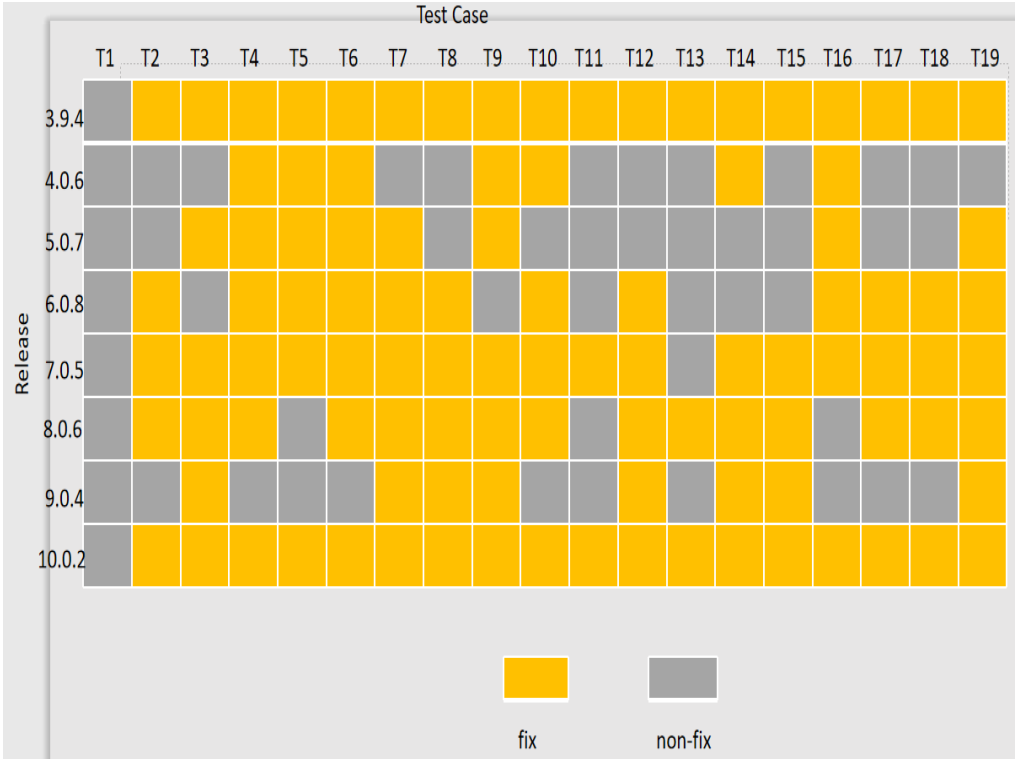


Figure 4.16. Modifications needed on various versions of Dolibarr, with EyeAutomate

4.1.5 Summary of Dolibarr Change Types under EyeAutomate

In EyeAutomate, I mainly applied selenium commands and image recognition technology. Between successive versions, the various types of corrections that appear are as follows:

- ClickName Change

  It would occur when t locates an element via ClickName command value that functioned (led to e) in V, but no longer functions in V'. There are many causes of such a change, such as an element value modification or change to locate an element via ClickID command. For example (see Figure 4.17), T2 of V2.9.0 tries to locate the "Zip" "Town" text field in the application using Selenium commands, but they are altered

from ClickName “cp” to ClickName “zipcode”, ClickName “ville” to ClickName “town” for successive V3.9.0.

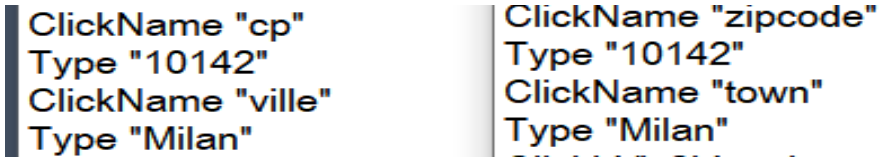


Figure 4.17. ClickName Change of Dolibarr, with EyeAutomate

- ClickID Change

  It would occur when t locates an element via ClickID command value that functioned (led to e) in V, but no longer functions in V’. There are many causes of such a change, such as an element value modification. For example (see Figure 4.18), T14 of V2.9.0 tries to locate the “Date start” text field in the application, but they are altered from ClickId “project” to ClickId “projectstart” for successive V3.9.4.

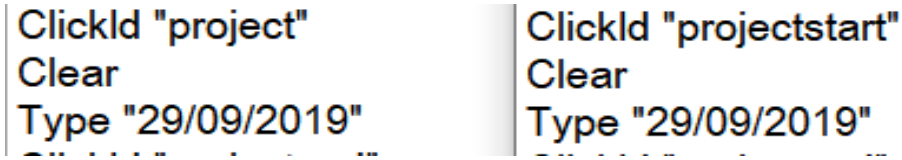


Figure 4.18. ClickID Change of Dolibarr, with EyeAutomate

- SelectText Change

  It would occur when t locates an element via its text from the drop-down list, using a text string that functioned (led to e) in V, but no longer functions in V’. For example (see Figure 4.19), in T4 of V2.9.0, There are two elements (Status (Sales), Nature) using ClickName and SelectText commands, but text content has changed from “Obsolete” “First Material” to “For sale” “Raw Material” separately. So I have to modify the text value for V3.9.4.

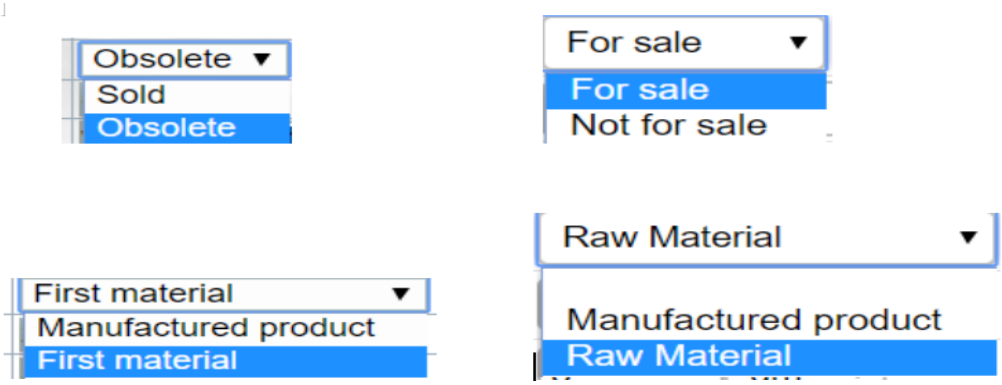


Figure 4.19. SelectText Modification of Dolibarr, with EyeAutomate

- Image Change

  EyeAutomate works well on any platform using intelligent image recognition. But sometimes the image is not recognized to a certain extent. So here I list the common situations that I met. The first (I set T9 of V4.0.6 as an example) is that the image has not changed, but cannot be identified in V5.0.7, and the image needs to reselect target position or region using the mouse (see Figure 4.20). The second (I set T18 of V2.9.0 as an example, see the left side of Figure 4.21) is that the content in the image has indeed changed, making it fail to run successfully in V3.9.4. The third scenario is that a web element can't continue to use the selenium command. At this time, it is better to use image recognition (see Figure 4.22).

Figure 4.20. Images Reposition of Dolibarr, with EyeAutomate

Figure 4.21. Image Substitution of Dolibarr, with EyeAutomate

Figure 4.22. Command-Image Conversion of Dolibarr, with EyeAutomate

- ClickText Change

  It would arise in a situation When t attempts to locate a link l via a *href* attribute value that functioned (led to l) in V, but no longer functions in V'. It could be link text change, link deletion, addition, etc. For example (see Figure 4.23), T8 of V2.9.0 attempts to locate a link via the content "Customer's order", but the link is deleted in V3.9.4. Attempt to locate a link via the content "Customer", but the link is deleted in V3.9.4. Attempt to locate a link via the content "Add order", but the link is deleted in V3.9.4.

```
ClickText "Customer's orders"
ClickText "New order"
ClickText "Customer"
ClickText "Add order"
ClickName "ref_client"
Type "000005"
```

```
ClickText "New order"
ClickName "ref_client"
Type "000005"
```

Figure 4.23. ClickText Change of Dolibarr, with EyeAutomate

- ClickCss Change

  It would occurs when t locates an element e via Css that functioned (led to e) in V, but no longer functions in V'. There are many causes of such a change, such as an ancestor of e in DOM tree modification, addition, deletion. For example (see Figure 4.24), T9 of V2.9.0 attempts to locate "Create draft" button, Css value is altered in V3.9.4 from ".button:nth-child(2)" to "#id-right > div > form > div.center > input".

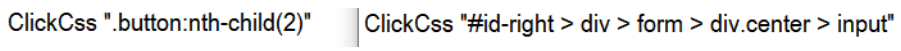


Figure 4.24. ClickCss Change of Dolibarr, with EyeAutomate

- SelectValue Change

It would occur when t locates an element e via its value from the drop-down list, using a value string that functioned (led to e) in V, but no longer functions in V'. For example, T11 in V6.0.8 (see Figure 4.25) attempts to locate "Payment type" via ClickId "selectpaymenttype" then select the option via SelectValue command. For the second command, it is altered from SelectValue "6" to SelectValue "LIQ" for successive V7.0.5 (see Figure 4.26).

```
▼<tr>
  ▼<td>
    ▼<span class="fieldrequired">
       <label for="selectpaymenttype">Payment type</label>
     </span>
    </td>
  ▼<td>
    ▼<select id="selectpaymenttype" class="flat selectpaymenttypes" name="paymenttype"> == $0
       <option value="0" selected> </option>
       <option value="2">Bank transfer</option>
       <option value="4">Cash</option>
       <option value="7">Cheque</option>
       <option value="6">Credit card</option>
       <option value="3">Debit payment order</option>
     </select>
     <img src="/dolibarr/theme/eldy/img/info.png" alt title="You can change values for this
     list from menu Setup - Dictionaries" class="hideonsmartphone">
   </td>
</tr>
```

Figure 4.25. SelectValue Modification1 of Dolibarr, with EyeAutomate

```
▼<tr>
  ▼<td>
    ▼<span class="fieldrequired">
       <label for="selectpaymenttype">Payment type</label>
     </span>
   </td>
  ▼<td>
    ▼<select id="selectpaymenttype" class="flat selectpaymenttypes" name="paymenttype">
       <option value selected> </option>
       <option value="VIR">Bank transfer</option>
       <option value="LIQ">Cash</option>
       <option value="CHQ">Cheque</option>
       <option value="CB">Credit card</option>
       <option value="PRE">Debit payment order</option>
     </select>
     <img src="/dolibarr/theme/eldy/img/info.png" alt title="You can change values for this list from menu
     Setup - Dictionaries" class="hideonsmartphone"> == $0
   </td>
</tr>
```

Figure 4.26. SelectValue Modification2 of Dolibarr, with EyeAutomate

- Element Deletion

In one case, an element in the Dolibarr webpage exists in the current version and does not exist in subsequent versions. Therefore, this element must be removed to fit the version being tested. For example (see Figure 4.27), T3 of V2.9.0 has an element “Prefix” but not exist in V3.9.4.

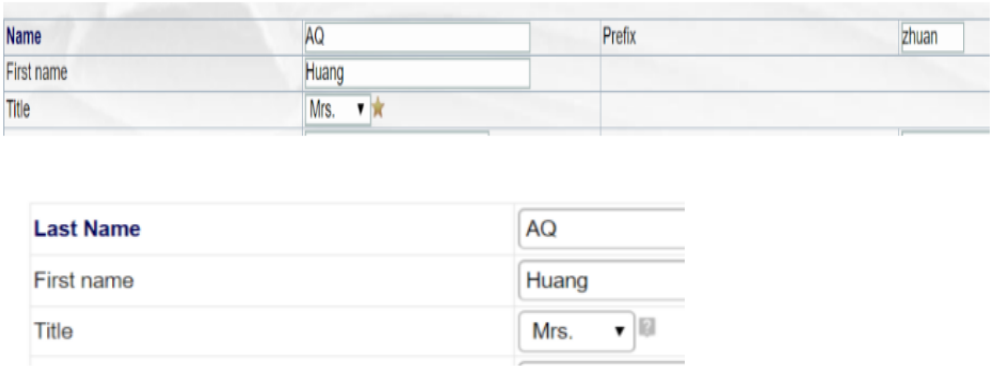


Figure 4.27. Element Deletion of Dolibarr, with EyeAutomate

- Element addition

There is a situation here, some already exist web elements (even here newly added elements in successive version), once it is not filled, then when you click the submit button, the application will notify you with an alert message. So these elements need to be added to our test case (see Figure 4.28 and 4.29), element “Date of payment” “Account” “Payment type” doesn’t exist in V2.9.0, but must be added to test case of V3.9.4.

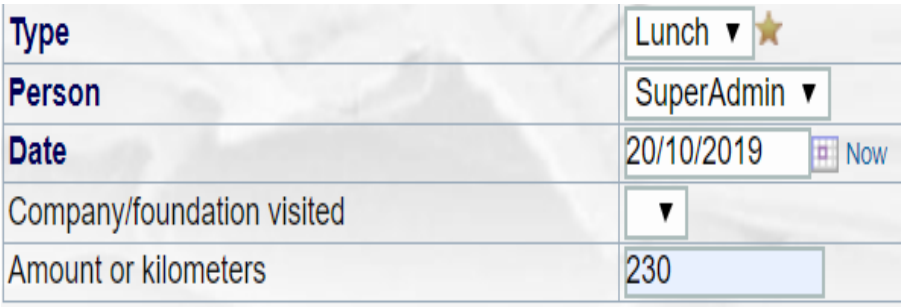


Figure 4.28. Element addition of Dolibarr in V2.9.0

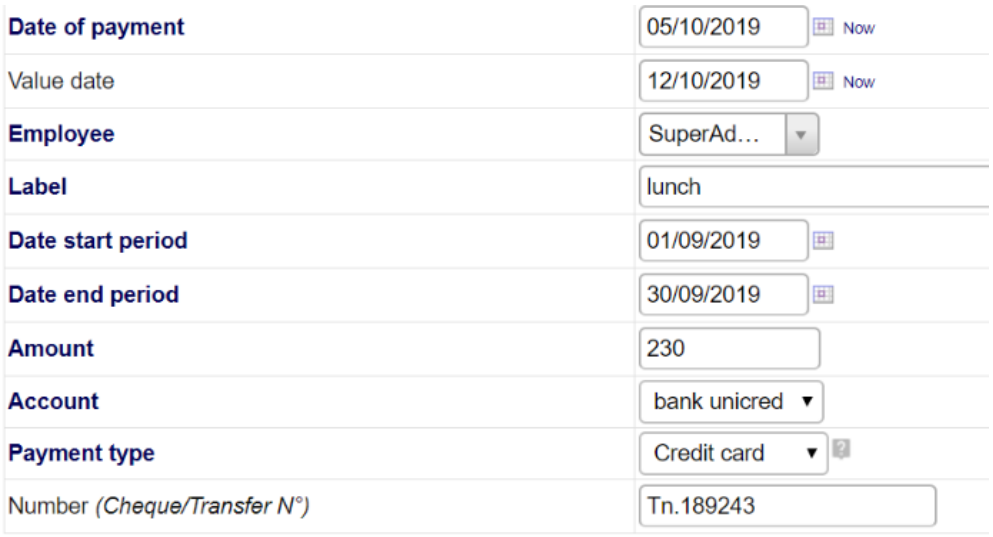


Figure 4.29. Element addition of Dolibarr in V3.9.4

4.1.6 Report on Dolibarr Test Cases under EyeAutomate

- Dolibarr V3.9.4

  All of the test cases of V2.9.0 needs to fix on this version of the app, except T1: the authentication test.

Table 4.9. Report on V3 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T2,T3,T4,T5,T6,T8,T9,T10,T11,T12,T13,T14,T15,T17,T19
ClickID Change	T2,T14
SelectText Change	T4,T5,T6,T12
Image Change	T2,T3,T4,T5,T6,T8,T9,T10,T12,T14,T15,T16,T17,T18,T19
ClickText Change	T8,T11,T14
ClickCss Change	T3,T9
SelectValue Change	NULL
Element Deletion	T3,T4,T5,T6,T8,T10,T11,T13,T15,T19
Element addition	T3,T11,T15,T18

- Dolibarr V4.0.6

  Seven tests out of nineteen need to fix in this version of the app. All tests written for V3.9.4 are compatible with this version, except T4: create a

new product in products/services module and T5: create a new service in products/services module and T6: create a new warehouse in products/services module and T9: create a new intervention in commercial module and T10: create a new donation in financial module and T14: create a new project in projects module and T16: set up member types in members module.

Table 4.10. Report on V4 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	NULL
ClickID Change	NULL
SelectText Change	T6
Image Change	T4,T5,T14
ClickText Change	T10,T16
ClickCss Change	T9
SelectValue Change	NULL
Element Deletion	NULL
Element addition	NULL

- Dolibarr V5.0.7

Eight tests out of nineteen need to fix in this version of the app. All tests written for V4.0.6 are compatible with this version, except T3: create a new third party in third party module and T4: create a new product in products/services module and T5: create a new service in products/services module and T7: create a new contact in commercial module and T6: create a new warehouse in products/services module and T9: create a new intervention in commercial module and T16: set up member types in members module and T19: create a new action/task in agenda module.

Table 4.11. Report on V5 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T16
ClickID Change	NULL
SelectText Change	T6,T19
Image Change	T3,T4,T5,T7,T9
ClickText Change	NULL
ClickCss Change	NULL
SelectValue Change	NULL
Element Deletion	NULL
Element addition	NULL

- Dolibarr V6.0.8

  Twelve tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T3: create a new third party in third party module and T9: create a new intervention in commercial module and T13: create a new bank account number in bank/cash module and T14: create a new project in projects module and T15: create a new task in projects module are completely compatible with the ones written for release V5.0.7.

Table 4.12. Report on V6 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T2,T12,T16
ClickID Change	NULL
SelectText Modification	T6
Image Change	T2,T4,T5,T6,T7,T8,T10,T12,T16,T17,T18,T19
ClickText Change	T2
ClickCss Change	NULL
SelectValue Modification	NULL
Element Deletion	NULL
Element addition	NULL

- Dolibarr V7.0.5

  Seventeen tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T13: create a new bank

account number in the bank/cash module are completely compatible with the ones written for release V6.0.8.

Table 4.13. Report on V7 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T2
ClickID Change	NULL
SelectText Change	NULL
Image Change	T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T14,T15,T16,T17,T18,T19
ClickText Change	T2
ClickCss Change	T10,T11
SelectValue Change	T11
Element Deletion	NULL
Element addition	NULL

- Dolibarr V8.0.6

  Fifteen tests out of nineteen need to fix in this version of the app. T1: User cannot log in due to invalid credentials and T5: create a new service in products/services module and T11: create new trips and expenses in the financial module and T16: set up member types in members module are completely compatible with the ones written for release V7.0.5.

Table 4.14. Report on V8 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T2
ClickID Change	NULL
SelectText Change	NULL
Image Change	T2,T3,T6,T7,T8,T9,T10,T12,T14,T15,T17,T18,T19
ClickText Change	T2,T3,T10,T13,T14
ClickCss Change	NULL
SelectValue Change	NULL
Element Deletion	T15
Element addition	NULL

- Dolibarr V9.0.4

  Eight tests out of nineteen need to fix in this version of the app. All tests written for V8.0.6 are compatible with this version, except T3: create a new third party in third party module and T7: create a new

contact in commercial module and T8: create a new customer's order in commercial module and T9: create a new intervention in commercial module and T12: create a new financial account in bank/cash module and T14: create a new project in projects module and T15: create a new task in projects module and T19: create a new action/task in agenda module.

Table 4.15. Report on V9 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	T3
ClickID Change	NULL
SelectText Change	NULL
Image Change	T3,T7,T8,T9,T12,T15
ClickText Change	T14
ClickCss Change	NULL
SelectValue Change	NULL
Element Deletion	NULL
Element addition	NULL

- Dolibarr V10.0.2

  Eighteen tests out of nineteen need to fix in this version of the app. Only T1: User cannot log in due to invalid credentials is completely compatible with the ones written for release V9.0.4.

  In this version, there are two special points that need to be pointed out.

  1. Once we open the webpage of the application, a toolbar will always appear at the bottom of the webpage, which prevents the script from running smoothly, so we need to eliminate this toolbar by clicking the cancel button. This situation occurs at T2, T3, T4, T5, T7, T8, T10, T11, T12, T13, T14, T15, T16, T17, T19.
  2. In this version, we must scroll the long web pages, otherwise the script will fail to run. This situation occurs at T2, T4, T6, T7, T9, T10, T11, T12, T13, T14, T16, T19.

  In addition to the above, there is a report about the V10 of the Dolibarr experiment result under EyeAutomate.

Table 4.16. Report on V10 of Dolibarr, with EyeAutomate

Situation	Occurred Test Case
ClickName Change	NULL
ClickID Change	NULL
SelectText Change	NULL
Image Change	T3,T5,T7,T8,T9,T14
ClickText Change	NULL
ClickCss Change	NULL
SelectValue Change	NULL
Element Deletion	T3,T5
Element addition	NULL

4.1.7 Comparison between Selenium and EyeAutomate

In Figure 4.30 I show the percentage of modified test cases for the whole test suite for each version of Dolibarr (see the last column) and the percentage of modified test cases for the whole corresponding test cases among versions of Dolibarr (see the last row). It works with Selenium. I do the same with EyeAutomate in Figure 4.31.

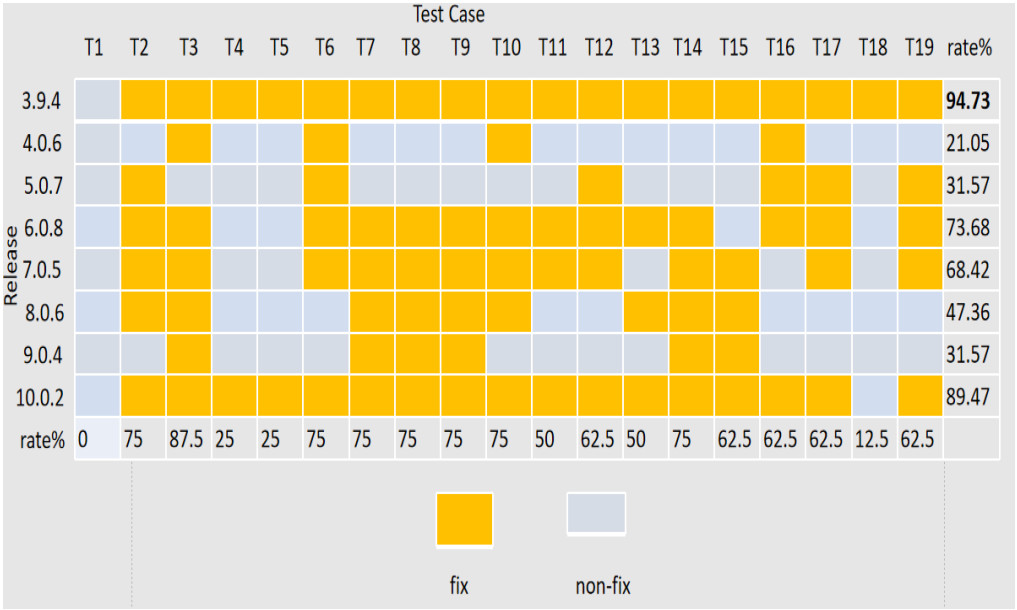


Figure 4.30. Percentage of Modified Tests with Selenium

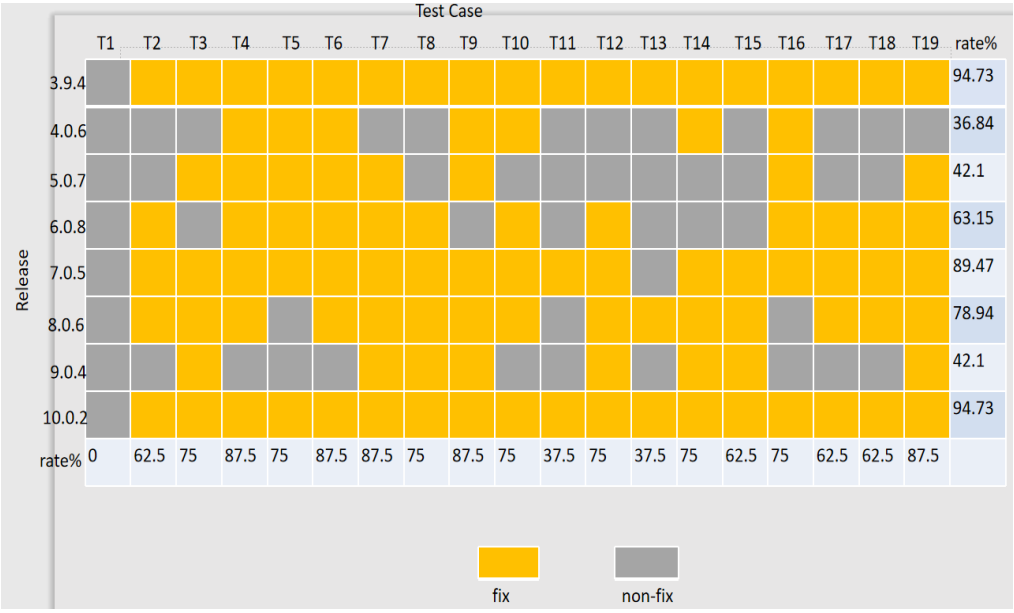


Figure 4.31. Percentage of Modified Tests with EyeAutomate

The data gathered on the application Dolibarr in this test report largely mirror that applying Selenium automated testing tool is much more stable than EyeAutomate. EyeAutomate testing tool bases on image recognition of web elements within the user interface of the web application. According to the experiment result, Tests created using EyeAutomate is fragile, because simple changes (the size and the color of the web element) to visual appearance could cause fragile testing.

In Table 4.17 and Table 4.18, It shows that some of the Lines of Code (LOC) of a test case have changed from version N-1 to version N of Dolibarr (The changes include deletion, addition, modification). For example, T2 applying Selenium changed 9 line codes in V2 into 11 line codes in V3. T19 applying EyeAutomate didn't fix line codes/instructions/commands from V3 to V4. So T19 is compatible between v3 and v4.

Table 4.17. Count changed LOCs with Selenium for Dolibarr

	V2>V3	V3>V4	V4>V5	V5>V6	V6>V7	V7>V8	V8>V9	V9>V10
T1	0->0	0->0	0->0	0->0	0->0	0->0	0->0	0->0
T2	9->11	0->0	1->1	3->3	8->7	6->6	0->0	0->9
T3	21->24	1->1	0->0	2->2	4->3	1->1	2->2	12->12
T4	12->13	0->0	0->0	0->0	0->0	0->0	0->0	2->8
T5	11->11	0->0	0->0	0->0	0->0	0->0	0->0	5->6
T6	12->13	2->2	3->3	3->3	4->3	0->0	0->0	0->5
T7	49->29	0->0	0->0	8->8	12->12	1->1	1->1	1->6
T8	18->16	0->0	0->0	2->2	3->3	1->1	1->1	0->1
T9	7->8	0->0	0->0	4->4	7->6	1->1	1->1	2->6
T10	18->18	1->1	0->0	2->2	5->4	1->1	0->0	2->10
T11	19->21	0->0	0->0	2->2	6->5	0->0	0->0	0->7
T12	16->17	0->0	1->1	7->8	15->10	0->0	0->0	0->1
T13	5->4	0->0	0->0	0->3	0->0	1->1	0->0	1->10
T14	8->8	0->0	0->0	1->1	5->4	1->1	1->1	1->5
T15	1->1	0->0	0->0	0->0	4->2	3->1	1->1	1->2
T16	2->4	1->1	1->1	2->2	0->0	0->0	0->0	1->4
T17	18->19	0->0	1->1	2->2	4->3	0->0	0->0	11->17
T18	1->3	0->0	0->0	0->0	0->0	0->0	0->0	0->0
T19	23->16	0->0	1->1	5->5	7->4	0->0	0->0	0->6

Table 4.18. Count changed instructions with EyeAutomate for Dolibarr

	V2>V3	V3>V4	V4>V5	V5>V6	V6>V7	V7>V8	V8>V9	V9>V10
T1	0->0	0->0	0->0	0->0	0->0	0->0	0->0	0->0
T2	4->8	0->0	0->0	4->3	6->5	4->4	0->0	0->2
T3	14->15	0->0	1->1	0->0	3->5	2->2	3->3	11->6
T4	7->6	1->1	2->2	2->2	2->2	1->1	0->0	2->4
T5	6->5	2->2	2->2	2->2	2->2	0->0	0->0	6->5
T6	7->8	4->4	1->1	2->2	3->4	1->1	0->0	0->2
T7	39->21	0->0	1->1	3->3	7->8	7->7	3->3	4->6
T8	12->10	0->0	0->0	3->5	5->5	3->3	1->1	2->3
T9	5->6	2->2	1->1	0->0	3->3	2->2	1->1	3->5
T10	10->10	1->1	0->0	2->1	5->5	1->1	0->0	0->4
T11	7->14	0->0	0->0	0->0	3->3	0->0	0->0	0->3
T12	6->12	0->0	0->0	7->6	7->7	4->5	1->1	0->2
T13	3->2	0->0	0->0	0->0	0->0	1->1	0->0	0->2
T14	5->5	2->2	0->0	0->0	1->1	1->1	1->1	1->3
T15	2->3	0->0	0->0	0->0	5->5	4->2	1->1	0->1
T16	1->1	1->1	1->1	2->2	1->1	0->0	0->0	0->3
T17	9->11	0->0	0->0	1->2	1->2	1->1	0->0	5->6
T18	4->7	0->0	0->0	4->4	1->1	3->3	0->0	2->2
T19	10->9	0->0	1->1	1->1	5->5	1->1	1->1	3->7

4.2 Fragility of Dolibarr

This section reports the detected defects and fragilities during the execution of the test cases on all the available releases of the Dolibarr based on Selenium and EyeAutomate testing.

One non-deterministic test failure was found by executing the developed test suites under Selenium testing, related to issues in performing a scroll operation on the web page of the app (example: Test report of T2 in V10.0.2 pointed out that Exception in thread “main” org.openqa.selenium.ElementClickInterceptedException: element click intercepted: Element <input name=“capital” id=“capital” class=“minwidth100” value=“ ”> is not clickable at point (632, 551). Other element would receive the click: ...). It occurs mostly

in v10.0.2 and only T13 in v6.0.8. It is spotted that the element is not clickable anymore, click on another position coordination. Once I add a scroll screen operation to the element, Selenium could recognize the element. According to the report of the EyeAutomate experiment part, this situation also occurred here.

Based on the analysis of test fragilities induced by the evolution, as discussed in the Selenium testing experiment result, I classified the causes of the fragilities that were observed in the study.

Identifier change: When the test case is invalidated by changes in one of the element attributes.

Text change: An element is not provided with a unique identifier, but is based on the text detected by their textual description. Text-based locator is fragile because a single different character is sufficient to invalidate a text-based locator based on String comparison.

Deletion or addition: Between two consecutive releases of the same application, an element may be removed or added.

Structure of DOM tree change: The fragilities are caused when the class or type of the elements change between two consecutive releases of the same application. Many locators were based on the name of the class of the elements, that was subject to change in different releases.

Based on the experimental results of the EyeAutomate test, I introduced the other causes of the fragilities observed in the study, which is derived from the taxonomy defined in one article, [35].

Graphic Change: As stated before, image recognition testing techniques applied in the EyeAutomate testing tool. When a new arrangement of the elements in the graphical layout - or even just a simple modification in the style of the application - this type of change in the Dolibarr is as expected as the primary cause of fragility for the Visual test suite. Examples of Graphic Change fragilities, related to the drop-down list option, are reported in Figure 4.32.

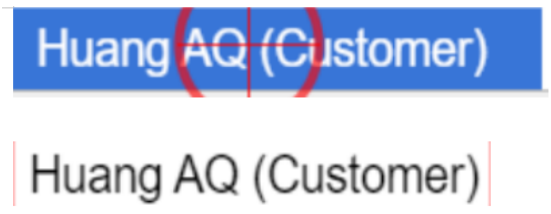


Figure 4.32. Example of Pure Graphic Fragilities

Widget Arrangement Change: This situation will occur when the relative position on the screen of the widgets to interact is changed between two different releases of the app. This fragility is experienced in one test case, which is reported as examples in Figure 4.33. In release 5.0.7 the arrangement of the widgets in a layout is changed.

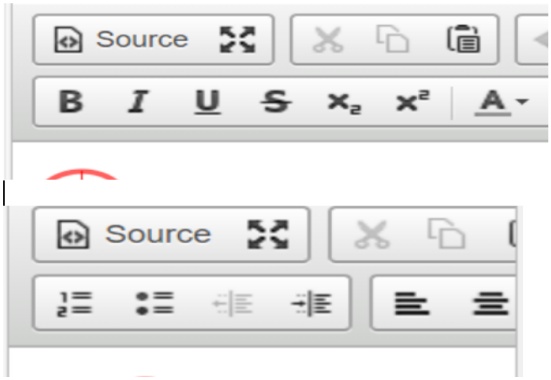


Figure 4.33. Example of Widget Arrangement Fragilities

Table 4.19 reports the main causes of the encountered fragilities during the execution of Selenium test cases. It also reports the percentage of maintenance in each version. For example, (see 14/19 in the table) there are 14 of 19 test cases that occur identifier modification in V3.9.4 compared with V2.9.0.

Table 4.19. Main causes of fragilities in broken test cases

Cause	V3.9.4	V4.0.6	V5.0.7	V6.0.8	V7.0.5	V8.0.6	V9.0.4	V10.0.2
Identifier change	14/19	1/19	1/19	2/19	13/19	2/19	1/19	0/19
Text change	5/19	3/19	2/19	2/19	1/19	6/19	2/19	0/19
Deletion or Addition	12/19	0/19	0/19	0/19	0/19	1/19	0/19	1/19
Structure of DOM tree change	14/19	0/19	4/19	12/19	13/19	3/19	3/19	5/19

Starting from the fragility found when adapting the test cases to different versions of the same application, I give some indications about practices

leading to fragility, that may be useful for the creation of guidelines for programmers aimed at avoiding such problems for the testing phase.

Chapter 5

Conclusion and Future Work

The primary objective of this research was to do automated testing of web applications with a variety of automated testing tools and discuss the fragility of the web application.

Firstly, I analyzed 4 distinct web application automated testing tools and practiced them on the Polito website.

Then, I specifically selected a web application, Dolibarr, which is open-source on GitHub, and used Selenium, EyeAutomate to do detailed automated testing of their various versions. Between different versions, each test case is tested and verified by a systematic and iterative procedure, and various fragility events appearing during the test web application are also found.

The main objective of my exploration was to discover and identify the relative main causes and characteristics of fragility.

Moreover, according to the results of these experiments, give indications to testers about how to avoid or at least minimize applying fragility test codes when they do test definitions for the web applications.

As possible extensions, this thesis can be replicated on other software domain (e.g., mobile or hybrid mobile/web applications) to evaluate the fragility issue in typologies of applications that are similar to typical web ones.

As well, the experiment can be replicated on multiple web applications, possibly written with different technologies, to extend its transferability and external validity.

Bibliography

[1] Giuseppe A. Di Lucca&Anna Rita Fasolino (2006): *Testing Web-based Applications: The State of the Art and Future Trends*. Information and Software Technology 48(12), pp. 1172–1186, doi:10.1016/j.infsof.2006.06.006.

[2] A. Stout (2001): *Testing a Website: Best Practices*. The Revere Group.

[3] Selenium IDE, https://www.seleniumhq.org/projects/ide/, 2017

[4] Selenium WebDriver, http://www.seleniumhq.org/projects/webdriver/,2017.

[5] R. T. Futrell, L. I. Shafer, and D. F. Shafer, *Quality Software Project Management*, Prentice Hall PTR,2001

[6] A. Ahmed, *Software Testing as a Service*, Auerbach Publications, New York:2009

[7] G.J.M yers. *The Art of Software Testing*, John Willey& Sons,Inc, New York, USA, 1979.

[8] W. C. Hetzel, *The Complete Guide to Software Testing*, 2nd ed. Publication info: Wellesley, Mass.:QED Information Sciences,1988. ISBN:0894352423

[9] M. Kumari, A. Sharma and V. Kamboi, *Replacement of S/W Inspection with S/W Testing*, International Journal of Information Technology and Knowledge Management July-December 2009,Volume 2, No. 2,pp. 257-261

[10] S.Thummalapenta, S.Sinha, N.Singhania and S.Chandra, "*Automating Test Automation*," 34th International Conference on Software Engineering (ICSE), 2012.

[11] T. Kanstrén, "*A Review of Domain-Specific Modelling, Software Testing*," The Eighth International Multi-Conference on Computing in the Global Information Technology, 2013.

[12] A.Jain and S.Sharma, "*An Efficient Keyword Driven Test Automation Framework for Web Applications*," International Journal of Engineering Science&Advanced Technology, vol. 2, no. 3, pp. 600-604, 2012.

[13] S.H.Trivedi, "*Software Testing Techniques*," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 10, pp. 433-439, 2012.

[14] Z. Huang and L. Carter. *Automated Solutions: Improving the Efficiency of software Testing*, The International Association for Computer Information Systems(IACIS) Conference, (IACIS 2003)Las Vegas October 4,2003.

[15] S.Berner, R.Weber and R.K.Keller, *Observations and Lessons Learned From Automated Testing*, in Proceedings of the 27th International Conference on Software Engineering(ICSE '05), pp. 571-579, St. Louis, Mo, USA, May 2005

[16] J.Kasurinen, O.Taipale,and K.Smolander, *Software Test Automation in Practice: Empirical Observations,Advances in Software Engineering*,vol. 2010,Article ID 620836,18 pages,2010.doi:10.1155/2010/620836.

[17] M. Leotta, A. Stocco, F. Ricca, and P. Tonella. *Reducing web test cases aging by means of robust xpath locators*. In Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pages 449–454, Nov 2014.

[18] Brett Daniel, Qingzhou Luo, Mehdi Mirzaaghaei, Danny Dig, Darko Marinov, and Mauro Pezzè. 2011. *Automated GUI Refactoring and Test Script Repair*. In Proceedings of First International Workshop on End-to-End Test Script Engineering (ETSE '11). ACM, 38–41.

[19] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. 2016. *Approaches and Tools for Automated End-to-End Web Testing*. Advances in Computers 101 (2016), 193–237.

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

[21] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella. Visual vs. *DOM-based web locators: An empirical study*. In Proceedings of the International Conference on Web Engineering (ICWE), pages 322–340. Springer, 2014.

[22] A. Mesbah, A. van Deursen, and S. Lenselink. *Crawling ajax-based web applications through dynamic analysis of user interface state changes*. ACM Transactions on the Web (TWEB), 6(1):3:1–3:30, 2012.

[23] P. Montoto, A. Pan, J. Raposo, F. Bellas, and J. Lapez. *Automating navigation sequences in Ajax websites*. In Proceedings of the International Conference on Web Engineering (ICWE), volume 5648, pages 166–180. Springer, 2009.

[24] S. Bruning, S. Weissleder, and M. Malek, "*A fault taxonomy for service-oriented architecture*," in IEEE High Assurance Systems Engineering Symposium, 2007, pp. 367–368.

[25] J. H. Hayes, "*Building a requirement fault taxonomy: Experiences from a NASA verification and validation research project*," in International Symposium on Software Reliability Engineering, 2003, pp. 49–60.

[26] L. Mariani, "*A fault taxonomy for component-based software*," in International Workshop on Test and Analysis of Component-Based Systems, 2003, pp. 55–65.

[27] K. S. M. Chan, J. Bishop, J. Steyn, L. Baresi, and S. Guinea, "*A fault taxonomy for web service composition*," in Service-Oriented Computing - ICSOC 2007 Workshops, ser. Lecture Notes in Computer Science, E. Di Nitto and M. Ripeanu, Eds. Springer, 2009, vol. 4907, pp. 363–375.

[28] W. Hummer, C. Inzinger, P. Leitner, B. Satzger, and S. Dustdar, "*Deriving a unified fault taxonomy for event-based systems*," in ACM International Conference on Distributed Event-Based Systems, Jul. 2012, pp. 167–178.

[29] F. Ocariza, K. Pattabiraman, and B. Zorn, "*JavaScript errors in the wild: An empirical study*," in International Symposium on Software Reliability Engineering, 2011, pp. 100–109.

[30] F. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah, "*An empirical study of client-side JavaScript bugs*," in International Symposium Empirical Software Engineering and Measurement, 2013, pp. 55–64.

[31] F. Ricca and P. Tonella, "*Web testing: A roadmap for empirical research*," in International Symposium on Web Site Evolution, 2007.

[32] A. Marchetto, F. Ricca, and P. Tonella, "*Empirical validation of a web fault taxonomy and its usage for fault seeding*," in International Workshop on Web Site Evolution, 2007, pp. 31–38.

[33] Hammoudi,Mouna, "*Why Do Record/Replay Tests of WebApplication Break?*"(2016). Computer Science and Engineering: Theses, Dissertations, and Student Research.100.http://digitalcommons.unl.edu/computerscidiss/100

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

[35] R. Copola, M. Morisio, and M. Torchiano. 2018. *Maintenance of Android Widget-Based GUI Testing: A Taxonomy of Test Case Modification Causes*. In 2018 IEEE International Conference on Software

Testing, Verification and Validation Workshops (ICSTW). 151-158. https://doi.org/10.1109/ICSTW.2018.00044