Comparing the Maintainability of Selenium WebDriver Test Suites Employing Different Locators: A Case Study

Maurizio Leotta, Diego Clerissi, Filippo Ricca, Cristiano Spadaro

Abstract:

Test suite maintenance tends to have the biggest impact on the overall cost of test automation. Frequently modifications applied on a web application lead to have one or more test cases broken and repairing the test suite is a time-consuming and expensive task.

This paper reports on an industrial case study conducted in a small Italian company investigating on the analysis of the effort to repair web test suites implemented using different UI locators (e.g., Identifiers and XPath).

The results of our case study indicate that ID locators used in conjunction with LinkText is the best solution among the considered ones in terms of time required (and LOCs to modify) to repair the test suite to the new release of the application.

Digital Object Identifier (DOI):

http://dx.doi.org/10.1145/2489280.2489284

http://dx.doi.org/10.1145/2489280.2489284

Comparing the Maintainability of Selenium WebDriver Test Suites Employing Different Locators: A Case Study

Maurizio Leotta[1], Diego Clerissi[1], Filippo Ricca[1], Cristiano Spadaro[2]

[1] DIBRIS - Università di Genova, Italy [2] eXact learning solutions S.p.A., Sestri Levante, Italy

{ maurizio.leotta | diego.clerissi | filippo.ricca }@unige.it, c.spadaro@exactls.com

ABSTRACT

Test suite maintenance tends to have the biggest impact on the overall cost of test automation. Frequently modifications applied on a web application lead to have one or more test cases broken and repairing the test suite is a time-consuming and expensive task.

This paper reports on an industrial case study conducted in a small Italian company investigating on the analysis of the effort to repair web test suites implemented using different UI locators (e.g., Identifiers and XPath).

The results of our case study indicate that ID locators used in conjunction with LinkText is the best solution among the considered ones in terms of time required (and LOCs to modify) to repair the test suite to the new release of the application.

Categories and Subject Descriptors:
D.2.5 [**Testing and Debugging**]: Testing tools

General Terms: Experimentation, Measurement

Keywords: Web Application Testing, Test Automation, Selenium WebDriver, Maintainability, UI Locators

1. INTRODUCTION

Importance of test automation in Web engineering is before everybody's eyes [5, 6]. Test Automation is considered crucial for the success of large web applications: it saves a lot of time in testing and helps to release web applications with fewer defects [1]. One of the main advantages of test automation is that software developers can run tests more often and finding bugs on the early stage of development.

However, "everything has a prize". In the context of automated web testing, the prize is associated with the concept of fragile tests: a *fragile test* is a test that can easily break when the Web Application Under Test (WAUT) changes. And, actually this happens very often because functional tests are usually built on top of web pages (e.g., recover a button in the web page, click it, verify a value in the created table) with the consequence that also a small change in a web page (e.g., in the layout of the page) leads to have one or more test cases broken (e.g., test cases are not more able to locate a link, an input field or a submission button). This implies that: when the web pages are frequently updated, the corresponding test suite has to be frequently repaired. This is a tedious and time consuming task for the test teams and an expensive task for the organizations since it has to be manually performed by software testers.

This paper reports some empirical observations and the challenges of a mixed test team of academy and industry workers new to Test Automation. The work was performed in close cooperation with the Software Engineering research group at Genoa University[1]. This case study started as a thesis project. During this exploratory case study, the objective of our industrial partner[2] was: comparing different solutions and ways to build automated tests for a web-based Learning Content Management System named eXact learning LCMS (one of the software they produce) and understanding which of them is able to reduce the maintenance effort needed to repair a broken test suite.

After an initial planning step [4], the test team decided to use the Selenium WebDriver framework and the *page object* pattern for producing the test cases. Thus, in this work we focus on the analysis of the costs associated with the repairing of Selenium WebDriver test suites implemented using different methods to locate UI elements (e.g., Identifiers and XPath). To the best of our knowledge, this is the first work trying to quantify the maintenance effort required to repair test suites adopting different locators.

The remainder of this paper describes the case study and its outcomes. Section 2 describes some useful "ingredients" to understand the case study: Selenium WebDriver, the *page object* pattern and some locators; Section 3 details the case study; Section 4 sketches the outcomes of the case study and presents the threats to validity. Finally, Section 5 concludes the paper.

2. SELENIUM "INGREDIENTS"

In this Section, we first explain Selenium WebDriver, i.e., the framework used to produce our test suite. Second, we introduce the *page object* pattern. Third, we present the various methods used in the Selenium WebDriver framework to locate UI elements. Finally, we show how to create and repair the test cases and the page objects using these methods.

[1] http://softeng.disi.unige.it/

[2] eXact learning solutions S.p.A., an Italian software development company: http://www.exact-learning.com/en/

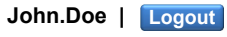
JAMAICA '13, July 15, 2013, Lugano, Switzerland


```
<form name="loginform" action="homepage.asp" method="post">
 Username: <input type="text" id="UID" name="username"><br>
 Password: <input type="text" id="PW" name="password"><br>
 <a href="javascript:loginform.submit()" id="login">Login</a>
</form>

<form name="loginform" action="homepage.asp" method="post">
 Username: <input type="text" name="username"><br>
 Password: <input type="text" name="password"><br>
 <a href="javascript:loginform.submit()">Login</a>
</form>
```

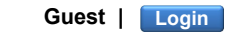
Figure 1: login.asp – Page & Source (with/without ID)

To make the description clearer, we will use a running example to explain the various concepts introduced. Let's imagine having to test a portion of a web application that allows to authenticate users. In a very simplified case, we could have a login page (called login.asp) similar to the one shown in Fig. 1 (top) that requires the users to enter the credentials (i.e., username and password). This page can be implemented in HTML as shown in Fig. 1, where two different versions are reported, with (middle) and without (bottom) ID attributes[3] for the HTML tags. For simplicity and to make the code clearer, we have omitted the tags not relevant for the comprehension of the example. After having inserted the credentials and clicked on "Login" the application shows the home page (homepage.asp). If the credentials are correct, the username and the logout button are reported in the upper right corner of the home page (see Fig. 2, top). Otherwise, the Guest user and the login link are shown (see Fig. 2, bottom). For simplicity, the application does not report any error message in case of invalid credentials or unrecognised users.




```
<div id="username">John.Doe</div> |
<a href="logout.asp" id="logout">Logout</a>

<div>John.Doe</div> | <a href="logout.asp">Logout</a>
```




```
<div id="username">Guest</div> |
<a href="login.asp" id="login">Login</a>

<div>Guest</div> | <a href="login.asp">Login</a>
```

Figure 2: homepage.asp (portion) as User (top) and as Guest (bottom) – Page & Source (with/without ID)

2.1 Selenium WebDriver

Selenium[4] is one of the most popular suites for automating web application testing and it is employed in many industrial projects [2, 3]. Essentially, this suite is composed of two tools: Selenium IDE[5] and Selenium WebDriver[6]. The former tool is implemented as Firefox extension and provides the recording and replay capabilities (i.e., Capture/Playback [7]). In practice, Selenium IDE records the user actions that can be transformed in test cases adding some assertions. Finally, test cases can be re-executed inside the Firefox browser. On the contrary, Selenium WebDriver provides a more comprehensive programming interface used to control a browser. Test cases are implemented manually in a programming language integrating Selenium WebDriver commands with JUnit or TestNG assertions. In this work we will focus on Selenium WebDriver, the framework chosen by our industrial partner.

2.2 Page Object

The *page object* pattern is used to model the web pages involved in the test process as objects, employing the same programming language used to write the test cases. In this way, the functionalities offered by a web page become "services" (i.e., methods) offered by the corresponding page object and they can easily be called within any test case. Thus, all the details and mechanics of the web page are encapsulated inside the page object. Adopting the *page object* pattern allows the test developer to work at a higher level of abstraction. The *page object* pattern is used to reduce the coupling between web pages and test cases. For these reason, adopting the *page object* pattern improves test suite maintainability [4].

2.3 UI Locators

Selenium WebDriver offers several different ways to locate the UI elements composing a web page. The most efficient one, according to Selenium WebDriver developers[7], is searching by their ID values (e.g., locate the password input field, in our example, by searching the value PW among the ID values). In case the ID attributes are not inserted in the HTML tags a XPath locator can be alternatively used (e.g., locate the password input field using the following XPath expression: /html/body/form/input[2]). Finally, the LinkText locator allows to select a hyperlink in a web page making use of its displayed text (i.e., the string contained between <a ...> and). Note that, since this method can be only used to locate links, it cannot be used to build a complete test suite. For this reason, it is only used as a complement of other locators[8].

2.4 Examples of Selenium Test Cases

Now, let us see how it is possible to test the above mentioned user authentication example employing Selenium WebDriver and using different UI locators. In all the test cases, we will use the *page object* pattern. The Java code has been slightly simplified to make the description more concise.

As an example, we report two simple test cases: a successful authentication test case and an unsuccessful one. The first logs in using correct credential (i.e., existing username and password: John.Doe and 123456) and verifies that in the home page the user has been authenticated (see Fig. 2, top). The second test case inserts unrecognised credentials and verifies that in the home page no user has been authenticated (i.e., Guest must be displayed in the upper right corner of the home page, see Fig. 2, bottom).

The first step is to create the two page objects LoginPage.java and HomePage.java corresponding to the web pages login.asp and homepage.asp (see Fig. 3). The page object LoginPage.java

[3] ID attributes specify unique identifiers for HTML elements (i.e., the value must be unique within the entire HTML document).

[4] http://seleniumhq.org/

[5] http://seleniumhq.org/projects/ide/

[6] http://seleniumhq.org/projects/webdriver/

[7] http://seleniumhq.org/docs/03_webdriver.jsp

[8] For the sake of completeness, Selenium WebDriver offers also other locators not considered here (e.g., CSS locators).

```
public class LoginPage {
 private final WebDriver driver;
 public LoginPage(WebDriver driver) {this.driver = driver;}
 public HomePage login(String UID, String PW) {
  driver.findElement(By.id("UID")).sendKeys(UID);
  driver.findElement(By.id("PW")).sendKeys(PW);
  driver.findElement(By.id("login")).click();
  return new HomePage(driver);
 }
}

public class HomePage {
 private final WebDriver driver;
 public HomePage(WebDriver driver) {this.driver = driver;}
 public String getUsername() {
  return driver.findElement(By.id("username")).getText;
 }
}
```

Figure 3: Page objects LoginPage.java and HomePage.java (ID locators version)

offers a method to log in in the application. That method takes as input a username and a password, inserts them in the corresponding input fields, clicks the Login button and returns a page object of kind HomePage.java (since after clicking the Login button the application moves to the page homepage.asp). HomePage.java contains a method that returns the username authenticated in the application or Guest when no user is authenticated. In these page objects, we have used the values of the ID attributes to locate the HTML tags.

The second step is to develop the two test cases making use of those page objects (see Fig. 4). More in detail, we can see that in each test method: first, a WebDriver of type FirefoxDriver is created allowing to control the Firefox browser[9]; second, the WebDriver (i.e., the browser) opens the specified URL and creates the page object, that is an instance of LoginPage.java, related to the page login.asp; third, using the method login(...) offered by the page object, a new page object (HP) representing the page homepage.asp is created; finally, the assertion can be checked using the method getUsername().

```
public void testLoginOK() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'login.asp' page
 driver.get("http://www.....com/login.asp");
 LoginPage LP = new LoginPage(driver);
 HomePage HP = LP.login("John.Doe","123456");
 // we are in the 'homepage.asp' page
 assertEquals("John.Doe", HP.getUsername());
 driver.close();
}

public void testLoginKO() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'login.asp' page
 driver.get("http://www.....com/login.asp");
 LoginPage LP = new LoginPage(driver);
 HomePage HP = LP.login("Inexistent","Inexistent");
 // we are in the 'homepage.asp' page
 assertEquals("Guest", HP.getUsername());
 driver.close();
}
```

Figure 4: TestLoginOK and TestLoginKO test cases

As said before, Selenium WebDriver offers also the possibility of locating hyperlinks by LinkText. Fig. 5 (top) presents an equivalent version of the method login(...) of the page object LoginPage.java where the LinkText locator is used.

There are different kinds of XPath expressions that can be used to locate elements. For instance, to locate the first input field in Fig. 1 (i.e., username), we could: (1) navigate the complete HTML tree starting from the beginning of the web

[9]Selenium allows also to employ other browsers, e.g., IE8.

```
public HomePage login(String UID, String PW) {
 driver.findElement(By.id("UID")).sendKeys(UID);
 driver.findElement(By.id("PW")).sendKeys(PW);
 driver.findElement(By.linkText("Login")).click();
 return new HomePage(driver);
}

public HomePage login(String UID, String PW) {
 driver.findElement(By.xpath("/html/body/form/input[1]")).sendKeys(UID);
 driver.findElement(By.xpath("/html/body/form/input[2]")).sendKeys(PW);
 driver.findElement(By.xpath("/html/body/form/a")).click();
 return new HomePage(driver);
}

public HomePage login(String UID, String PW) {
 driver.findElement(By.xpath("/html/body/form/input[1]")).sendKeys(UID);
 driver.findElement(By.xpath("/html/body/form/input[2]")).sendKeys(PW);
 driver.findElement(By.linkText("Login")).click();
 return new HomePage(driver);
}
```

Figure 5: LoginPage.java (ID+LinkText, XPath and XPath+LinkText locators versions)

page with the XPath expression /html/body/form/input[1], or (2) select the 1st element of type input using the expression (//input)[1]. Fig. 5 (middle) presents the method login(...) of the page object LoginPage.java built using the first format.

Finally, also XPath can be combined with the LinkText locator, obtaining the code shown in Fig. 5 (bottom).

2.5 Examples of Test Suite Repairing

As an example, in the following, we consider four equivalent test suites implemented using different UI locators (i.e., ID, ID+LinkText, XPath and XPath+LinkText) and three possible modifications of our Login page that break the test suites.

- In the login.asp page the "Login" link name changes in "Login Now!". The page objects LoginPage in the ID+LinkText and XPath+LinkText test suites need to be repaired (see Fig. 6). On the contrary, the other test suites are not affected by this change.
- The structure of the login.asp page is changed (e.g., the form is placed inside a table). Only the page objects LoginPage in the XPath and XPath+LinkText test suites need to be repaired. ID locators are indifferent to this change.
- The IDs of all the login.asp page elements are changed (it could happen, for instance, when the IDs are auto-generated in a "dumb" way, e.g., id1, ..., idn). Only the page objects LoginPage in the ID and ID+LinkText test suites need to be repaired.

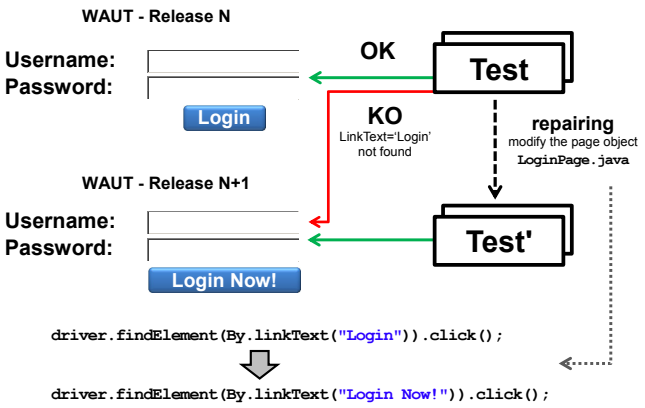


Figure 6: Repairing required when the"Login" link changes in "Login Now!"

3. THE CASE STUDY

The *goal* of this work is understanding which UI locator (among the ones seen before) is able to reduce the maintenance effort needed to repair a broken test suite of a WAUT.

3.1 Web Application Under Test

The Web Application Under Test is eXact learning LCMS. It is mainly a Learning Content Management System for eLearning content production that contains also a Learning Management System for eLearning content delivery. The product is a web application developed in ASP.NET that relies on a Microsoft SQL Server database. It is designed with a multi-tier approach consisting of presentation, business and data access layers. The development started about 6 years ago, with a development team composed by 3-4 software analysts and developers. eXact learning LCMS is currently composed by about 700.000 lines of code, 200 ASP.NET web pages, and has been developed using Visual Studio IDE. The developers of eXact learning LCMS used a facility of Visual Studio IDE that adds an auto-generated ID for each HTML tag (e.g., ctl00_Menu_Repeater_ctl01_Link).

3.2 Compared Test Suites

As a first step, the test team focused on a portion of the eXact learning LCMS web application to test with Selenium WebDriver. They chose the DURP portion, i.e., the portion that manages the Domains, Users, Roles and Permissions that can be defined in the application. The test team opted for that portion because it is crucial for an LCMS and because it is quite common in all the web-based applications (so it is possible to reuse the knowledge gained in this project and the produced test cases).

The test team developed a first Selenium WebDriver test suite using the ID locators and the *page object* pattern[10]. They chose to employ: (1) ID locators, since it is the most efficient method (see Section 2.3), taking advantage of auto-generated IDs, and (2) the *page object* pattern, since it allows to reduce the coupling between test cases and web pages.

The so developed test suite for the DURP portion of eXact learning LCMS is composed of 25 test cases and 19 page objects for a total of 3320 Java LOCs[11] (1720 for the test cases and 1600 for the page objects). Overall, there are 131 localization lines in the 19 page objects. This means that the 25 test cases use 131 lines in total to locate the UI elements. We recall that, a localization line is the line calling the driver to find a UI element (e.g., all the lines starting with `driver.findElement` in Fig. 3 are localization lines).

Each test case in our test suite performs several steps such as navigating web pages, clicking links, filling forms and evaluating assertions. They were built using conditional statements, loops, logging functionality, exception handling, reporting functionality and all of them are parameterized (a.k.a. data-driven) test cases[12].

As an example, we describe the AddUserTest test case. This test case has been developed to test the functionality that allows, only to certain roles (e.g., administrator), to add new users to the application. The test case opens the LoginPage, logs in with an administrator account (login and pwd are recovered from a CSV file) and navigates the HomePage, AdministrationPage, and UserMngPage to reach the AddNewUserPage. Then, the test case fills the form with the user data (recovered from a CSV file) and submits it. If everything is ok, the eXact learning LCMS application shows a summary page (called UserDetailsPage) listing the inserted user data. At this point, the test case locates each web page element displaying the data inserted (e.g., username, name, surname, email) and verifies that the values contained in the page are equal to the ones inserted before.

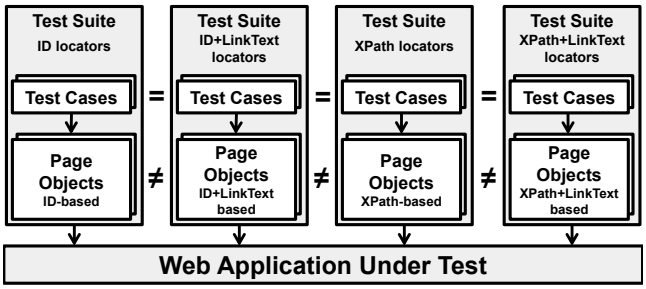


Figure 7: Test Suites, Test Cases and Page Objects

Note that the test suite is not trivial: when the test suite is executed, 336 test case instances are run (since each test case is parameterized with several different input/expected values stored in a CSV file). The complete execution of the test suite takes about 3 hours and half employing a computer equipped with an Intel Core i5 dual-core processor (3.1 GHz), 8 GB RAM and a fast (100 Mb/s) network connection to the servers hosting the eXact learning LCMS application.

Then, starting from the ID test suite, we built three equivalent test suites substituting the ID locators with XPath and LinkText locators obtaining the ID+LinkText, XPath and XPath+LinkText test suites. All these new test suites are equivalent to the ID one, since the test cases are exactly the same (they test the same functionalities using the same services offered by the page objects) and only the methods used to locate the UI elements inside the page objects are different (see Fig. 7). Note that, the effort for repairing a test case also includes the one required to repair the page objects it uses.

In our experiment, during the development of the XPath-based test suites, we chose to use absolute XPath expressions (e.g., /html/body/form/input[1]). Indeed, given that all the web page elements in the WAUT have an ID, locating them by means of relative XPath expressions (e.g., //*[@id='UID']) would be equivalent to find these elements using their IDs. Even if, at first sight, absolute XPath expressions could appear quite complex, we have used two tools to automatically build/modify them (i.e., FireBug[13] and FirePath[14]).

3.3 Research Question & Dependent Variable

We compared the four test suites trying to answer to the following research question:

[10]This test suite is a slightly modified version of the one used in [4].

[11]LOCs have been measured as the number of source code lines (without imports, comment lines or empty lines) formatted following the Sun's Java Code Conventions. http://java.sun.com/docs/codeconv/index.html

[12]Parameterized test cases are test cases executed several times, each time passing them different arguments (i.e., input and expected values).

[13]https://addons.mozilla.org/en/firefox/addon/firebug/

[14]https://addons.mozilla.org/en/firefox/addon/firepath/

RQ: *Which is the best locator among ID, ID+LinkText, XPath and XPath+LinkText in terms of maintenance cost reduction?*

In other words, we are interested to determine whether one (or more) of the proposed methods for locating UI elements is clearly better than the other ones for reducing the effort to repair a test suite when a new version of a WAUT is created. We measured the dependent variable *repairing effort* in terms of time (minutes) and number of line of code to change for all the four considered test suites.

3.4 Repairing Procedure

The four equivalent test suites were developed for the release M9 of eXact learning LCMS. During the 2012 a new release (M10) was developed and the four test suites were no longer working on it.

To reach our goal (i.e., quantifying the effort needed to repair the four test suites and understanding which is the best locator), we had to measure the time required to repair the test suites. Since, we conducted a case study with only two software testers working in pair programming (one from the industry and one from the academy) and not a controlled experiment with several participants, it was necessary to devise a procedure able to reduce as much as possible any possible learning effect among the repairing tasks conducted on different test suites[15].

We decided to perform the repairing of each test suite every 7 days. In this way, we reduced possible learning effects, since after 7 days it is really difficult to remember which are the auto-generated IDs or XPath expressions to modify. Moreover, it is important to highlight that the modifications required to repair a test suite can be partially or totally different among different test suites since each test suite employs a different way to locate UI elements. This is particularly true among the ID and XPath based test suites. For this reason, to distance as much as possible similar test suites, we ordered the repairing tasks in the following way: ID test suite repairing, 7 days pause, XPath test suite repairing, 7 days pause, ID+LinkText test suite repairing, 7 days pause, XPath+LinkText test suite repairing.

Finally, for each test suite, the two software testers ordered in the same way the test cases composing them. For all the test cases, the two software tester executed the algorithm described in Fig. 8 to repair them.

```
Procedure 'repairing #n'{
  res = run testcase;
  if (res == PASSED) then goto repairing #(n+1);
  else{                  //res == ERROR or FAILURE
    record start time;       //(e.g., 10:24)
    try
      {realign testcase;}
    catch (new-BUG-found-in-the-application){
      abort repairing #n;
      goto repairing #(n+1);
    }
    record stop time;        //(e.g., 10:56)
  }
}
```

Figure 8: Test case repairing algorithm ($1 \leq n \leq 25$)

[15]Note that, instead the learning that we could have during the repairing of different test cases in the same test suite is not a problem since this is natural in a real context (e.g., the repairing of the test #2 is affected by the learning originated by the repairing of test #1).

4. RESULTS

As a first step, we ran the four test suites against the new release of eXact learning LCMS (M10), observing that all the 25 test cases composing each test suite failed for all the four test suites. This means that all the considered test suites are equally fragile. Then, for each test suite, we applied the repairing procedure above explained, noting down: (1) the time required to repair each test case and (2) the number of LOCs really modified. During the repair task, we discovered that the changes made in eXact learning LCMS (M10) did not effect the logic of the test cases (i.e., the modifications were minimal; only the layout/structure of the pages was changed). For this reason, we modified only the page objects, and more precisely a subset of the 131 lines implementing the localization of UI elements (see Section 3.2).

Fig. 9 shows, by means of a scatterplot, the time required to repair the test cases to the new release of eXact learning LCMS. The test cases using ID and ID+LinkText locators are represented by means of rectangles, while test cases using XPath and XPath+LinkText locators are represented by means of triangles. The four regression curves shown in Fig. 9 have been computed using the distance-weighted least squares method of STATISTICA[16].

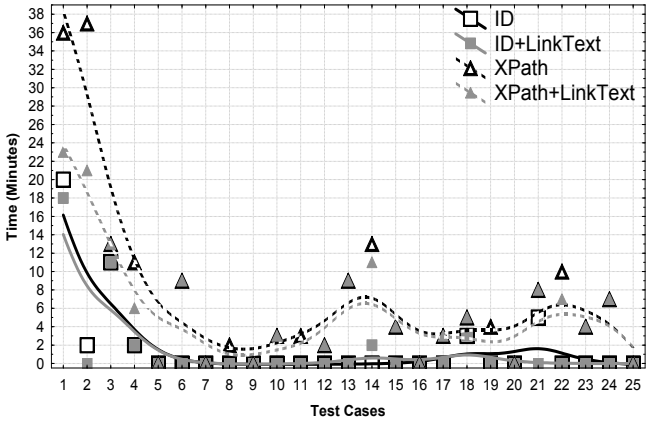


Figure 9: Time to repair the test cases

From Fig. 9 it is evident that the time spent for repairing XPath-based test cases goes beyond the time for repairing the ID-based ones in almost all the cases. Moreover, it is interesting to note that the trend of the time required for repairing the test cases is similar for all the four test suites (high for the first test cases and low for the subsequent ones, see the regression curves in Fig. 9). The first test cases required more time to be repaired than the subsequent ones since the *page object* pattern has been adopted. Indeed, to repair the first test cases of each test suite (e.g., test cases 1, 2, 3 and 4), we modified the page objects they use. But, these modifications were also useful for the subsequent test cases (e.g., 5, 6, and 7), thus they required less time to be repaired.

Fig. 10 globally summarizes our data in terms of total repairing time and number of modified LOCs. The two ID-based test suites have required a similar amount of repairing time, 43 minutes when the ID locator is used and 36 minutes

[16]A polynomial (second-order) regression is calculated for each value on the X variable scale to determine the corresponding Y value: an algorithm similar to the one used here is described by McLain, 1974.

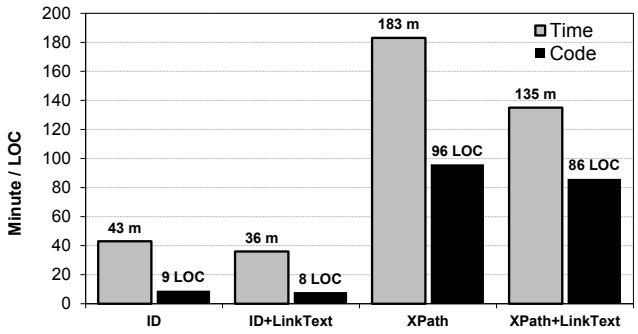


Figure 10: Time required and LOCs modified

when it is used in conjunction with the LinkText locator. This is correlated with the number of localization lines modified that are very similar, 9 and 8 LOCs respectively. Thus, we can say that: (1) repairing the two ID-based test suites required a similar amount of time and LOCs to modify, and (2) when LinkText is used we noted a slightly improvement for both variables (time –16.28%[17] and LOCs –11.11%). Instead, the two XPath-based test suites have decisively obtained worse results. Indeed, for their repairing, they required 183 minutes with the XPath locator and 135 minutes with the XPath+LinkText locator. Also for what concern the number of LOCs to modify (i.e., the localization lines) it is significantly higher than ID-based test suites, since the modified LOCs are 96 and 86 LOCs respectively. This is correlated with the high number of localization lines modified, i.e., 96 over 131 (73.28%) when the XPath locator is used by itself and 86 over 131 (65.65%) when the LinkText locator is used in conjunction with XPath. Also for XPath-based test suites, the combined usage with the LinkText locator gave an advantage in terms of time and LOCs to modify (time –26.23%[18] and LOCs –10.42%).

The answer to our research question RQ can be easily deduced by Fig. 11. It summarizes some results about the four implemented test suites: ID-based test suites have required the lowest maintenance effort, while the XPath-based test suites have required a higher effort (about 4 times more for what concerns the time and 10 times more for the number of LOCs to modify). In both cases the combined use of ID and XPath with LinkText has slightly improved the results.

4.1 Threats to Validity

Several factors could have influenced the results of our case study. First, we considered only a portion (the DURP) of a specific software for the Learning Content Management domain. Second, we considered a limited set of changes between two subsequent releases. However, it is important to underline that the eXact learning LCMS application is a real industrial application and that the considered changes are the typical ones that can be found during the development/maintenance of any web application (e.g., bug fixing, addition of web pages and layout modification). Thus, although the project involved the testing of specific software for the Learning Content Management domain, the experience is likely to be applicable across many commercial and government domains and we believe that the obtained results could also be generalized to other types of web applications.

[17]Computed using the eq.: $43–43x=36$

[18]Computed using the eq.: $183–183x=135$

	Time		Code	
Test Suites	**Minutes**	**Difference**	**LOCs**	**Difference**
ID	43		9	
ID+LinkText	36	- 16 %	8	- 11 %
XPath	183	+ 326 %	96	+ 967 %
XPath+LinkText	135	+ 214 %	86	+ 856 %

Figure 11: Comparison of the effort required when different locators are used

Future replications involving different applications and more expert software testers would help in generalizing the results. Finally, learning effects may have occurred while the testers repaired the test suites even if we tried to limit them as much as possible with a specific procedure.

5. CONCLUSION AND FUTURE WORK

The main result of our industrial case study is that, from the point of view of repairing effort, ID locators are better than XPath locators even if IDs are auto-generated. Moreover, the combined use of ID or XPath with LinkText has slightly improved the results with respect to the bare locators. These results are not conclusive, but this work compares for the first time, in a real industrial context, four Selenium test suites built using different UI locators and confirm with some data the anecdotal knowledge that locating elements by identifier is better. Moreover, these results, if confirmed, could help a manager interested in testing activities to make important decisions. For example, if the WAUT has to be maintained for a long time and meaningful identifiers have not been added to the web elements during development, it is better adding auto-generated IDs to the web elements and using the ID locators instead of XPath locators.

Already planned future works, will be devoted to a more extended study on the actual benefits of ID locators. In particular, we would like to extend our automated test suite to cover the entire eXact LCMS application.

6. REFERENCES

[1] S. Berner, R. Weber, and R. Keller. Observations and lessons learned from automated testing. In *Proc. of ICSE 2005*, pages 571–579. IEEE, 2005.

[2] E. Collins and V. de Lucena. Software test automation practices in agile development environment: An industry experience report. In *Proc. of AST 2012*, pages 57–63. IEEE, 2012.

[3] B. Haugset and G. Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Proc. of AGILE 2008*, pages 27–38. IEEE, 2008.

[4] M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro. Improving test suites maintainability with the page object pattern: an industrial case study. In *Proc. of 6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013)*, page (to appear). IEEE, 2013.

[5] F. Ricca and P. Tonella. Testing processes of web applications. *Ann. Softw. Eng.*, 14(1-4):93–114, 2002.

[6] F. Ricca and P. Tonella. Detecting anomaly and failure in web applications. *IEEE MultiMedia*, 13(2):44–51, 2006.

[7] T. Wissink and C. Amaro. Successful test automation for software maintenance. In *Proc. of ICSM 2006*, pages 265–266. IEEE, 2006.