Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study

Maurizio Leotta, Diego Clerissi, Filippo Ricca, Cristiano Spadaro

Abstract:

The page object pattern is used in the context of web testing for abstracting the application's web pages in order to reduce the coupling between test cases and application under test. This paper reports on an industrial case study in a small Italian company (eXact learning solutions S.p.A.) investigating the potential benefits of adopting the page object pattern to improve the maintainability of Selenium WebDriver test cases. After a maintenance/evolution activity performed on the application under test, we compared two equivalent test suites, one built using the page object pattern and one without it. The results of our case study indicate a strong reduction in terms of time required (by a factor of about three) and number of modified LOCs (by a factor of about eight) to repair the test suite when the page object pattern is used.

Digital Object Identifier (DOI):

http://dx.doi.org/10.1109/ICSTW.2013.19

Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study

Maurizio Leotta[1], Diego Clerissi[1], Filippo Ricca[1], Cristiano Spadaro[2]
[1] Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), Università di Genova, Italy
[2] eXact learning solutions S.p.A., Sestri Levante, Italy
maurizio.leotta@unige.it, diego.clerissi@gmail.com, filippo.ricca@unige.it, c.spadaro@exactls.com

***Abstract*—The page object pattern is used in the context of web testing for abstracting the application's web pages in order to reduce the coupling between test cases and application under test. This paper reports on an industrial case study in a small Italian company (eXact learning solutions S.p.A.) investigating the potential benefits of adopting the page object pattern to improve the maintainability of Selenium WebDriver test cases. After a maintenance/evolution activity performed on the application under test, we compared two equivalent test suites, one built using the page object pattern and one without it. The results of our case study indicate a strong reduction in terms of time required (by a factor of about three) and number of modified LOCs (by a factor of about eight) to repair the test suite when the page object pattern is used.**

***Keywords*—Web Application Testing, Test Automation, Selenium WebDriver, Test Suite Maintainability, Page Object Pattern.**

I. Introduction

The importance of test automation in web engineering is evident considering the number of companies investing in automated testing frameworks and tools nowadays. 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], [4]. The main advantage of test automation comes from being able to quickly run a set of tests after some changes have been made to a web application.

Selenium[1] is one of the most popular suites for automating web application testing and it is employed in many industrial projects. Essentially, this suite is composed of two tools: Selenium IDE[2] and Selenium WebDriver[3]. The former tool is implemented as Firefox extension and provides recording and replay capabilities (i.e., Capture/Playback [5]). In practice, Selenium IDE records the user actions that can be transformed into test cases by adding some assertions. Finally, test cases can be re-executed inside the Firefox browser. On the contrary, the latter tool provides a more comprehensive programming interface used to control a browser. Test cases are manually implemented in a programming language (e.g., Java) integrating Selenium WebDriver commands with JUnit or TestNG assertions.

This paper reports empirical observations and the challenges of a mixed test team of two academic and industry workers new to test automation and Selenium. The work was performed in close cooperation with the Software Engineering research group at Genoa University[4]. This case study started as a thesis project.

During this case study, the objectives of our industrial partner[5] were: (i) selecting a tool for test automation, (ii) implementing some automated test cases for a Web-based Learning Content Management System named eXact learning LCMS (one of the products they produce) and, (iii) starting to experiment and quantify potential benefits of the final adopted solution.

Finally, the test team resorted to Selenium WebDriver. They produced a test suite for a portion of eXact learning LCMS composed of 25 test cases written using the *page object* pattern[6]. This pattern is becoming popular in test automation for enhancing the maintainability of test cases.

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, even if further experimentation is needed. To the best of our knowledge, this is the first work trying to quantify the actual benefits of the *page object* pattern adoption in an industrial context.

The remainder of this paper describes the case study and its outcomes, and draws some conclusions on the benefits of using Selenium WebDriver in conjunction with the *page object* pattern. The paper is structured as follows: Sect. II details the case study; Sect. III describes the *page object* pattern; Sect. IV describes the realignment[7] procedure; Sect. V sketches the outcomes of the case study and discusses them. Sect. VI reports related work. Finally, Sect. VII concludes the paper.

II. The Case Study

The Web Application Under Test (WAUT) is the eXact learning LCMS web application. eXact learning LCMS is mainly a Learning Content Management System (LCMS) for eLearning content production that contains also a Learning Management System (LMS) 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

[1] http://seleniumhq.org/
[2] http://seleniumhq.org/projects/ide/
[3] http://seleniumhq.org/projects/webdriver/
[4] http://softeng.disi.unige.it/
[5] eXact learning solutions S.p.A. – http://www.exact-learning.com/en/
[6] http://code.google.com/p/selenium/wiki/PageObjects
[7] In this work, we use *realign* as synonym of *repair*.

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.

As a first step, the test team selected a portion of the eXact learning LCMS web application to test with Selenium. They chose the DURP portion of eXact learning LCMS, 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 application and because it is quite common for web-based applications.

Then, the test team started to produce some test cases using the recording capability of Selenium IDE. Quickly, the test team discovered several limitations in its usage. Natively, Selenium IDE does not provide some useful features, such as: conditional statements, loops, logging functionality, exception handling, and parameterized (a.k.a. data-driven) test cases[8].

Thus, the test team moved to Selenium WebDriver, but the problem became how to develop the test cases quickly, since WebDriver does not provide mechanisms to automatically record the test case. Thus, the test team decided to use Selenium IDE to produce a skeleton of the test cases and then they manually refined/refactored the exported Java scripts adding assertions, conditions and loops[9]. Moreover, when useful, the test team manually transformed the produced test cases into parameterized test cases. The result of this work was a Selenium WebDriver test suite (that we named PatternNO) for the DURP portion of eXact learning LCMS composed of 25 test cases and 2891 LOCs[10].

Each test case of the test suite performs several steps such as navigating web pages, clicking links, filling forms and finally evaluating a set of assertions. 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 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 the web page elements displaying the data inserted (e.g., username, name, surname, email) and verifies the correctness of these values.

By using this approach (i.e., combining Selenium WebDriver+IDE) the test team quickly produced the test cases but their quality was not sufficiently good. The produced Selenium test suite had a lot of duplicated code and the test cases were extremely coupled with the structure of the web pages. Surfing the Web, the test team discovered the *page object* pattern used precisely for addressing this kind of problems. Thus, they produced a new test suite (called PatternYES) completely equivalent to PatternNO but using the *page object* pattern. The production of the PatternYES test suite was more challenging and time-consuming with respect to the production of the PatternNO test suite[11] because it was completely conducted manually. The new test suite is composed of 25 test cases and 19 page objects for a total of 3320 LOCs (1720 for the test cases and 1600 for the page objects).

When each test suite is executed, 336 test case instances are run (since each test case is parameterized with several different input/expected values previously stored in a CSV file). The complete execution of each 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. To reach the best performances, the test team chose to locate the web page elements using their ID values, since, for localization purposes, this is the most efficient solution[12], and XPath expressions when ID values are not available.

Finally, we compared the two test suites (PatternNO and PatternYES) trying to answer to the following research question:

Does the adoption of the page object pattern reduce the effort needed to repair a Selenium test suite? and if yes, how much?

We measured the realignment effort in terms of time (minutes) and number of line of code to change for both the test suites.

III. The Page Object Pattern

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 can be easily called within any test case. Thus, all the details of the web page are encapsulated inside the page object. Adopting the *page object* pattern allows the test developer to work on a higher level of abstraction (clearly, except when it is necessary to develop the page objects).

In what follows, we show how it is possible to test a simple web application employing Selenium WebDriver with and without the *page object* pattern. Imagine having to test a portion of a web application that allows to book flights. In a very simplified case, we can have two pages like the ones

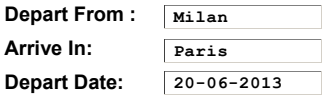
[8]Parameterized test cases are test cases executed several times, each time passing them different arguments (i.e., input and expected values).

[9]Note that the replay feature of Selenium IDE was not used.

[10]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

[11]Unfortunately the test team had not noted down the exact times.

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




```
<form name="input" action="departingFlights.asp" method="get">
  Depart From: <input type="text" id="orig" name="origAirport"><br>
  Arrive In:   <input type="text" id="dest" name="destAirport"><br>
  Depart Date: <input type="text" id="date" name="date"><br>
  <input type="submit" id="submit" value="Find Flights">
</form>
```

Fig. 1. searchFlights.asp – Page and Source (Version 1)

Departure		Arrival		Prices
Milan	08:30	Paris	09:45	89€
Milan	09:30	Paris	10:45	79€

```
<table id="results" border="1"><tr>
   <td colspan="2">Departure</td>
   <td colspan="2">Arrival</td>
   <td>Prices</td>
 </tr><tr>
   <td id="o1">Milan</td><td>08:30</td>
   <td id="d1">Paris</td><td>09:45</td><td>89€</td>
 </tr><tr>
   <td id="o2">Milan</td><td>09:30</td>
   <td id="d2">Paris</td><td>10:45</td><td>79€</td>
 </tr>
</table>
```

Fig. 2. departingFlights.asp – Page and Source

shown in Fig. 1 (called searchFlights.asp) and Fig. 2 (called departingFlights.asp). The first page allows the user to enter departure, destination and date of a flight. This page can be implemented in HTML as shown in Fig. 1 (for the sake of simplicity, we have omitted all the tags that are not relevant for the comprehension of the example). Fig. 2 shows the dynamically generated page containing all the found flights.

Now, let us see how it is possible to test the flight booking application. The following Java code has been slightly simplified to make the description more concise avoiding to report a lot of minor details. Without the *page object* pattern we can code a JUnit test class provided of test methods like the ones reported in Fig. 3 (i.e., testEuropeanFlight() and testNullFlight()). 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 as a real user does[13]; second, the WebDriver (i.e., the browser) opens the specified URL; third, the input fields are filled and the submission button is clicked (note that in this example the web elements are located using the values of their IDs); finally, when the submission button is clicked the WebDriver goes in another page (i.e., departingFlights.asp) and the assertions can be checked.

As it is possible to see in Fig. 3, the code implementing the test methods is very coupled to the web pages' implementation. For instance, in the test cases code, IDs are used to identify the various input forms and submission buttons. The adoption of the *page object* pattern allows to insert a level of abstraction

[13]Selenium WebDriver allows to employ also several other browsers.

```
public void testEuropeanFlight() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'searchFlights.asp' page
 driver.get("http://www.....com/searchFlights.asp");
 driver.findElement(By.id("orig")).sendKeys("Milan");
 driver.findElement(By.id("dest")).sendKeys("Paris");
 driver.findElement(By.id("date")).sendKeys("20-06-2013");
 driver.findElement(By.id("submit")).submit();
 // we are in the 'departingFlights.asp' page
 for each flight X returned, X = [1..n] {      //(o1,d1)...(on,dn)
  assertEquals("Milan", driver.findElement(By.id("oX")).getText());
  assertEquals("Paris", driver.findElement(By.id("dX")).getText());
 }
 driver.close();
}

public void testNullFlight() {
 WebDriver driver = new FirefoxDriver();
 driver.get("http://www.....com/searchFlights.asp");
 driver.findElement(By.id("orig")).sendKeys("Milan");
 driver.findElement(By.id("dest")).sendKeys("Milan");
 driver.findElement(By.id("date")).sendKeys("20-06-2013");
 driver.findElement(By.id("submit")).submit();
 assertEquals("Error",driver.findElement(By.id("msg")).getText());
 driver.close();
}
```

Fig. 3. Test Cases: SearchFlightsTest.java (without *page object* pattern)

```
public class SearchFlightsPage {
 private final WebDriver driver;
 public SearchFlightsPage(WebDriver driver) {this.driver = driver;}
 public DepartingFlightsPage searchFlights(String orig, String dest,
                                           Date date) {
  driver.get("http://www.....com/searchFlights.asp");
  driver.findElement(By.id("orig")).sendKeys(orig);
  driver.findElement(By.id("dest")).sendKeys(dest);
  driver.findElement(By.id("date")).sendKeys(date.toString());
  driver.findElement(By.id("submit")).submit();
  return new DepartingFlightsPage(driver);
 }
}
```

Fig. 4. Page Object: SearchFlightsPage.java

```
public void testEuropeanFlight() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'searchFlights.asp' page
 SearchFlightsPage SFP = new SearchFlightsPage(driver);
 DepartingFlightsPage DFP = SPF.searchFlights("Milan","Paris",
                            new Date(20,06,2013));
 // we are in the 'departingFlights.asp' page
 for each flight X returned, X = [1..n] {
  assertEquals("Milan", DFP.getOrigFlight(X));
  assertEquals("Paris", DFP.getDestFlight(X));
 }
 driver.close();
}

public void testNullFlight() {
 WebDriver driver = new FirefoxDriver();
 SearchFlightsPage SFP = new SearchFlightsPage(driver);
 DepartingFlightsPage DFP = SPF.searchFlights("Milan","Milan",
                            new Date(20,06,2013));
 assertEquals("Error", DFP.getMessage());
 driver.close();
}
```

Fig. 5. Test Cases: SearchFlightsTest.java (with *page object* pattern)

between the test cases and the web pages with the aim of reducing the coupling among them. In the following, we will see how the test cases change when using the *page object* pattern. The first step is to create a page object for each web page involved in the test cases. Here, for space reasons, we consider only the searchFlights.asp page (the other is conceptually analogous). The associated page object is SearchFlightsPage.java (see Fig. 4). It offers a method that returns a page object (an instance of the class DepartingFlightsPage.java) pertaining to the web page containing the found flights. The test cases reported in Fig. 3 can be simplified (see Fig. 5) making use of the methods provided by the page objects SearchFlightsPage.java and DepartingFlightsPage.java (not reported here). With the *page object* pattern it is not necessary to insert explicit references to the pages' implementation inside the test cases: all the pages' details are encapsulated inside the page objects.

Now, imagine that a change is made to our flight booking web application. For example, the input field used to enter the departure date changes from DD-MM-YYYY, to three fields: one for day, month, and year, respectively. Fig. 6 reports the appearance of the new searchFlights.asp page and the related HTML implementation.


```
<form name="input" action="departingFlights.asp" method="get">
  Depart From: <input type="text" id="orig" name="origAirport"><br>
  Arrive In:   <input type="text" id="dest" name="destAirport"><br>
  Depart Date:
  <input type="text" id="dateD" name="dateD">
  <input type="text" id="dateM" name="dateM">
  <input type="text" id="dateY" name="dateY"><br>
  <input type="submit" id="submit" value="Find Flights">
</form>
```

Fig. 6. searchFlights.asp – Page and Source (Version 2)

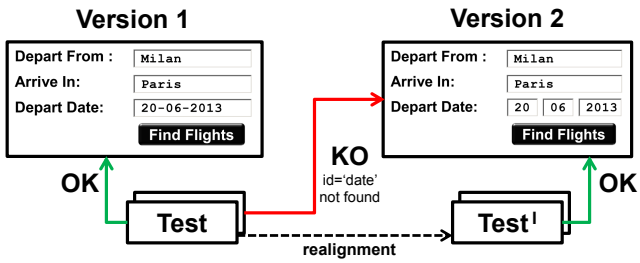


Fig. 7. Test Cases Fragility and Realignment

The previous test cases (see Fig. 3 and Fig. 5) do not work when they are executed on this new version of the web application, since they are unable to locate the element characterized by date as ID value (see Fig. 7).

Thus, the test cases need to be realigned to the new release of the web application. To this end, we have to replace the lines used to locate and insert the date in the input field. Fig. 8 (top) reports the line that has to be removed, followed by the new three lines that have to be inserted in the test cases not using the *page object* pattern. Note that in this case the modification has to be performed on both the tests method (i.e., testEuropeanFlight() and testNullFlight()). On the contrary, when adopting the *page object* pattern we have to apply a quite similar modification (see Fig. 8 (bottom)) but only in one place: inside the page object reported in Fig. 4.

```
driver.findElement(By.id("date")).sendKeys("20-06-2013");

driver.findElement(By.id("dateD")).sendKeys("20");
driver.findElement(By.id("dateM")).sendKeys("06");
driver.findElement(By.id("dateY")).sendKeys("2013");
```

```
driver.findElement(By.id("date")).sendKeys(date.toString());

driver.findElement(By.id("dateD")).sendKeys(date.getDay());
driver.findElement(By.id("dateM")).sendKeys(date.getMonth());
driver.findElement(By.id("dateY")).sendKeys(date.getYear());
```

Fig. 8. Test Cases Realignment without (top) and with (bottom) *page object* pattern

IV. Realignment Procedure

Recall that each test case in the PatternYES test suite was implemented starting from the corresponding test case in the PatternNO test suite. For this reason, each pair of test cases (PatternYES #n, PatternNO #n) tests a functionality of the eXact learning LCMS web application exactly in the same way. Thus, the difference between the two test suites is only in the usage of the *page object* pattern; for everything else they are perfectly equivalent.

The two test suites were developed for the release M9 of eXact learning LCMS. During the 2012 a new release (M10) was developed, and the two test suites were no longer working on the new release (i.e., all the 25 test cases composing both test suites fail or return an error when executed).

The goal of this case study is to quantify the effort needed to realign the two test suites using the following metrics: number of modified LOCs and time required to complete the realignment. To measure correctly the time required to realign a test suite, it was necessary to devise a procedure able to minimize any possible learning effect. Indeed, carrying out completely the task of realignment first on a test suite and after on the other one is not advisable. It would strongly affect the validity of the time recorded.

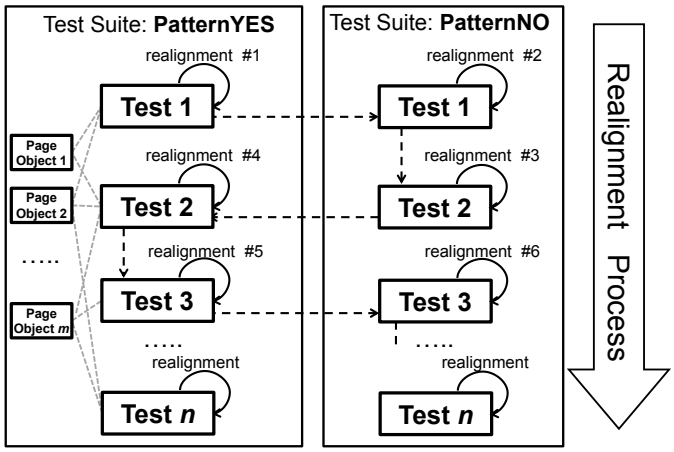


Fig. 9. Test suites Realignment Process

```
Procedure 'realignment #n'{
  res = run testcase;
  if (res == PASSED) then goto realignment #(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 realignment #n;
      goto realignment #(n+1);
    }
    record stop time;        //(e.g., 10:56)
  }
}
```

Fig. 10. Test case Realignment Procedure

The best choice would be to run a controlled experiment with a large number of software testers (for example using a completely randomized design [6]). In this way, each tester would have worked only on one version of the test suite (PatternYES or PatternNO), nullifying any possible learning effect[14].

Since, we are conducting a case study with only two software testers (one from the industry and one from the academy) and not a controlled experiment, we decided to distribute the learning, resulting from the knowledge gained during the realignment process, over both test suites. To reach this goal, the testers worked on the two test suites at the same time in pair programming. More in detail, the two software testers ordered in the same way the test cases composing the test suites. Thus, the test case #n in the PatternYES test suite, that tests a particular functionality of eXact learning LCMS (e.g., inserting a new user), corresponds to the test number #n in the PatternNO test suite. As described in Fig. 9, odd test cases were aligned first in the PatternYES version and then in the PatternNO one, while, on the contrary, the even test cases were aligned first in the PatternNO version and then in the PatternYES one. Alternating the two test suites has allowed us to distribute the knowledge gained during the process of realignment on both test suites thus minimizing (as much as possible) the learning effect on the overall realignment time. To realign each test case, the software testers followed the procedure described in Fig. 10.

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

V. RESULTS

This section reports the results from our experimental study. Finally, a qualitative discussion is presented.

As a first step, we ran the two test suites (built with and without page objects) against the new release of eXact learning LCMS (M10), observing that all the 25 test cases failed for both the test suites. No failures were due to faults/errors in the application but only to changes that made test cases broken. Then, for both the test suites, we applied the realignment procedure as explained above noting down the realignment time and the number of LOCs modified for each test case.

Fig. 11 shows, by means of a scatterplot, the time required to realign each test case to the new release of eXact learning LCMS. Test cases produced using the page object pattern (PatternYES) are represented by means of rectangles, while test cases produced without the page object (PatternNO) are represented with triangles. We remind that for PatternYES test cases the realignment time includes the time for modifying the test cases and the time for realigning the related page objects. The two regression curves shown in Fig. 11 have been computed using the distance-weighted least squares method of STATISTICA[15].

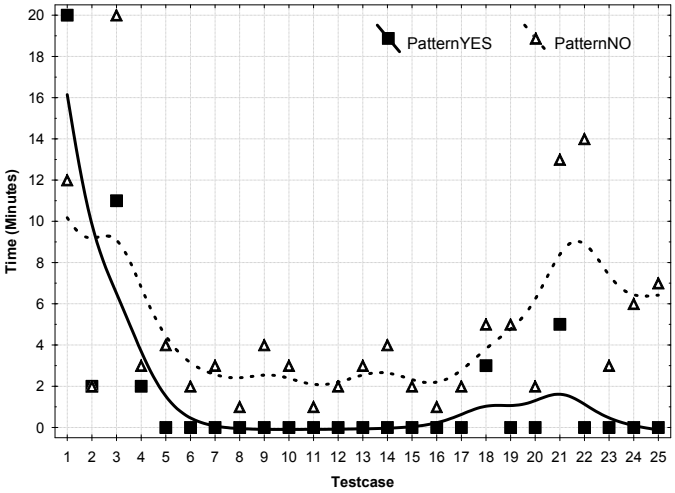


Fig. 11. Time of test cases realignment

From Fig. 11, it is evident that the time spent to realign the PatternNO test suite overcomes the time to realign the PatternYES ones in all the test cases except that for the test case 1 (see the regression curves). In this case, the time was greater (8 minutes more) because for realigning the PatternYES test case 1 was also necessary to modify three page objects. Test cases 2, 3 and 4 required to modify only one page object each. It is interesting to note that, from test case 5 to test case 15, the realignment time is continuously zero because no page objects and test cases had to be modified. Finally, only the test cases 18 and 21 have a realignment time greater than zero because they required to modify one page object respectively. Overall, the PatternYES test suite has required to modify six page objects (i.e., 11 LOCs) out of total 19 page objects without any modification of the test cases. On the contrary, the PatternNO test suite realignment has required to modify all the 25 test cases (i.e., 90 LOCs).

Fig. 12 globally summarizes our data in terms of total realignment time and total number of modified LOCs. The realignment time for the PatternNO test suite took 124 minutes in total vs. 43 minutes for the PatternYES test suite. It means 81 minutes less for the test suite produced using the page objects. In percentage, that value corresponds to a gain in terms of time of 65.32%[16]. This percentage is still larger in terms of modified LOCs: for the PatternNO test suite it was necessary to modify 90 LOCs in total to complete the realignment while only 11 LOCs were modified for the PatternYES test suite (reduction of modified LOCs = 87.77%[17]).

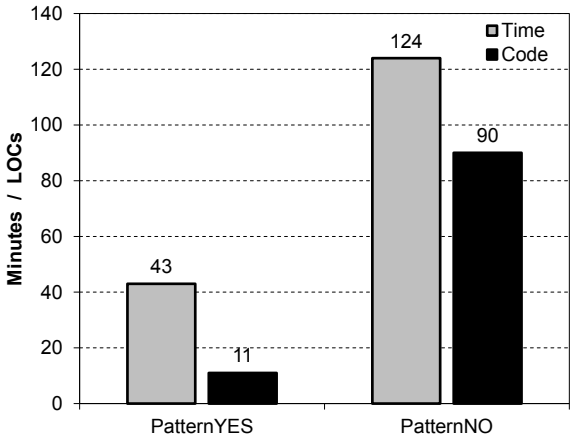


Fig. 12. Total realignment time and total number of modified LOCs

A. Discussion

The results of our experimental study, conducted in a real industrial setting, confirm the belief on the effectiveness of adopting the *page object* pattern to improve the maintainability of Selenium WebDriver test cases.

Thus, we can give a positive answer to our research question (see Section II): the adoption of the *page object* pattern reduces the effort needed to repair a Selenium WebDriver test suite. In particular, in our case study, the *page object* pattern has reduced: (i) by a factor of about three, the time required for realigning the test suite and (ii) by a factor of about eight, the number of LOCs to modify.

Furthermore, we observed that the adoption of the *page object* pattern:

– often concentrates the changes required to realign the test cases within the page objects, thus avoiding to modify the test cases. Indeed, after the maintenance intervention on eXact learning LCMS, we have not modified the test cases composing the PatternYES test suite but we have only changed the related page objects. However, in case of radical maintenance/evolution activities performed on the WAUT, it may be necessary to change also the test cases using the *page object* pattern; for instance when the navigation flow among

[15] A polynomial (second-order) regression is calculated for each value on the X variable scale to determine the corresponding Y value such that the influence of the individual data points on the regression decreases with their distance from the particular X value (an algorithm similar to the one used by this procedure is described by McLain, 1974).

[16] Computed using the eq.: 124–124*x*=43

[17] Computed using the eq.: 90–90*x*=11

the pages changes or when new input fields are added for data not required previously;

– does not reduce the fragility level of the test cases. In fact, when we ran the two test suites (PatternYES and PatternNO) against the new version of eXact learning LCMS, we observed that all the 25 test cases failed for both the test suites. This was expected since, the test cases composing the two test suites are "linked" to the web pages exactly in the same way (i.e., they locate the page elements using the same WebDriver commands), even if in the case of the PatternYES test suite the "linking" is done within the page objects;

– seems to reduce the productivity of software testers. In the case of the PatternYES test suite, modifying 11 LOCs required 43 minutes (i.e., 15.34 LOCs/hour) while in the PatternNO test suite, modifying 90 LOCs required 124 minutes (i.e., 43.55 LOCs/hour). However, this improvement is only apparent. In the case of the PatternNO test suite, the same corrections were repeated several times in different test cases (i.e., with fast copy and paste) and this has increased the number of modified LOCs per hour, but obviously not the real productivity of the testers.

Finally, an aspect that does not affect the obtained results of our experimental study, but that is very interesting and that deserves further investigation, emerges from the analysis of the reasons of why the tests failed on the new version of eXact learning LCMS (M10). Recall that, the test cases composing both test suites use, when possible, the ID values to locate the elements inside the web pages. During the process of realignment, we realized that the majority of the test cases' problems were due to modifications of the ID values in the M10 release of eXact learning LCMS. This is caused by the fact that a subset of the ID values in that application are strings concatenated with numbers generated automatically (e.g., id1, id2, id3, ... , idN). Thus, the addition of a new element (e.g., a new link) may change, in the web page structure, the IDs of the elements following the new introduced element and so broking the test cases using them.

VI. Related Work

Collins and de Lucena [2] describe their experience in test automation during the agile development (Scrum method) of a web application. Similarly to us, they built the automated test suite using Selenium IDE (our first tentative) and executed it using Selenium RC (while we used its successor, Selenium WebDriver). In the first phases of their project, they tried to automate the testing process as much as possible. However, as often happen at the beginning of a new project, the web pages were frequently updated because they were not able to meet users' needs. As a consequence, the test team had to re-record and re-write the test cases very often. In this way, the testing process was too time consuming, so they decided to limit the usage of automated test cases to only "stable" web pages, with the drawback of reducing the automated test suites coverage. We believe that the adoption of the *page object* pattern would have limited this problem.

Berner *et al.* [1] describe their experiences on testing automation, gained participating in dozen of industrial projects. Among the other things, they report that, since test automation is intended to save as much money as possible spent for "unproductive" testing activities, often companies expect to achieve a very fast return from their investments on test automation. However, if these expectations are not met, test automation is abandoned quickly. This fact may not provide the testers of enough time to reach the level of knowledge required to master and apply effectively techniques and solution like Selenium WebDriver and the *page object* pattern, thus preventing their adoption. For this reason, we plan to carry out further investigation aimed to evaluate the effort required to build automated test cases, using or not the *page object* pattern, and combine the results with the one reported in this work. In particular, we are interested to prove what already claimed in [1]: "maintenance tends to have a much bigger impact on the overall cost for testing than the initial implementation of automated tests". Positive results, emerging from our investigation, could encourage managers to go beyond the first phases of test automation adoption characterized by investments without immediate returns.

Finally, Karhu *et al.* [3] analysed empirically the factors affecting the adoption of software testing automation, conducting 30 interviews with managers, testers and developers from five different organizations. Also in this case, important factors were, on the one side, the time reduction (and the resulting reduction in personnel costs), but on the other side, the initial costs increment due to test cases implementation and personnel training.

VII. Conclusion and Future Work

The main result of our case study is that the adoption of the *page object* pattern has reduced, in a substantial way, the time required to realign our test suite (65.32% reduction) and the number of LOCs to modify (87.77% reduction). These results are not conclusive, but this work tries to quantify, for the first time, the actual benefits of the *page object* pattern in a real industrial context and illustrates several considerations about its usage.

Already planned future works, will be devoted to a more extended study on the actual benefits of the *page object* pattern. In particular, we would like to extend our automated test suite to cover the entire eXact learning LCMS application (we remind that in this work we have considered only the DURP portion). Another interesting direction is investigating the additional effort and cost (if any) required to model web pages with page objects during the process of test automation. Indeed, from a manager point of view, the adoption of the *page object* pattern should take into account also the costs it possibly introduces.

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.

[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.

[3] K. Karhu, T. Repo, O. Taipale, and K. Smolander. Empirical observations on software testing automation. In *Proc. of ICST 2009*, pages 201–209.

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

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

[6] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.