A Family of Experiments to Assess the Impact of Page Object Pattern in Web Test Suite Development

Maurizio Leotta*, Matteo Biagiola†, Filippo Ricca*, Mariano Ceccato‡, Paolo Tonella§
* Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), Università di Genova, Italy
† Fondazione Bruno Kessler, Trento, Italy
‡ Dipartimento di Informatica (DI), Università di Verona, Italy
§ Software Institute - Università della Svizzera italiana, Lugano, Switzerland
maurizio.leotta@unige.it, biagiola@fbk.eu, filippo.ricca@unige.it, mariano.ceccato@univr.it, paolo.tonella@usi.ch

***Abstract*—Automated web testing is an appealing option, especially when continuous testing practices are adopted. However, web test cases are known to be fragile and to break easily when a web application evolves. The Page Object (PO) design pattern addresses such problem by providing a layer of indirection that decouples test cases from the internals of the web page, where web page elements are located and triggered by the web tests. However, PO development could potentially introduce an additional burden to the already strictly constrained testing activities.**

This paper reports an empirical investigation of costs and benefits due to the introduction of the PO pattern in web test suite development. In particular, we conducted a family of controlled experiments in which test cases were developed with and without the PO pattern. While the benefits of POs did not compensate for the extra development effort they require in the limited experimental setting of our study, results indicate that when the test suite to be developed is at least 10× larger, test development becomes more efficient with than without POs.

***Index Terms*—End-to-End Testing, Web Testing, Family of Experiments, Page Object, Selenium Web Driver.**

I. Introduction

Testing web applications is a major challenge for software companies. Manual testing is labour-intensive, error-prone, and costly. It is also not compatible with agile development and continuous testing. Test automation frameworks and tools represent a valid alternative. Indeed, they have reached a high level of maturity and popularity, as in the case of Selenium WebDriver [1]. These tools usually work at the presentation level, interacting with the web elements that are visualized on the web page, as seen by the end-users. This kind of testing is called end-to-end (E2E) testing, because the application is tested as a whole system, under execution scenarios similar to the end-user ones: test cases submit inputs and receive outputs through the same web pages involved in end-user interactions.

With Selenium WebDriver, testers use a high-level programming language (e.g., Java) to develop test cases that consist of Selenium commands simulating the user's actions and retrieving information to verify the expected results. Then, test cases are executed in an unattended way by Selenium.

Even if test automation can increase the overall software quality [2], it comes with downsides. In fact, automated test cases are often fragile, i.e., when the web application evolves to accommodate requirements changes, test cases may easily break, and testers must correct them. This is especially true when test cases are strongly coupled with the web application under test [3]. For instance, a test case that locates a web page element in a list based on its index may no longer work if the position of such element in the list changes when the web application evolves.

Recently the *Page Object* (PO) design pattern has emerged as an important solution and best practice to address such test maintenance difficulties, reducing code duplication and lowering the coupling between test cases and the web application under test [4]. Although there is empirical evidence on the benefits associated with the adoption of the PO pattern during maintenance of web test suites [5], it is not clear if such adoption justifies the initial additional effort needed to implement the page objects (POs). Building POs for a web application is in fact a non-trivial activity, which may require substantial effort.

As a preliminary step to understand if the usage of POs pays off, we have designed and executed a family of three experiments with MSc and PhD students from different universities, with the goal of investigating the impact of the PO pattern on the development of web test cases. Since we are interested in comparing the implementation effort between PO-based test cases and non PO-based ones, our family of experiments has been designed with two treatments (with and without POs). To balance the experimental design for the learning effect, we used two different web applications (i.e., the objects of the experiments) and participants were asked to implement the test cases with each treatment in two consecutive laboratories using different objects. In our experiments, the test case development time increased significantly when POs were adopted, but such increment becomes negligible when the test suite size grows, because the methods provided by each PO are reused many times by different test cases. For test suites more than 10 times larger than those developed in our experiment (constrained to two hours of test case development), the adoption of POs is expected to give a positive return of the investment and test case development becomes more efficient with POs than without POs. It is reasonable to assume that such a test suite size is frequently reached in industrial settings.

The paper is organized as follows: Section II provides some background on the Page Object design pattern. Section III

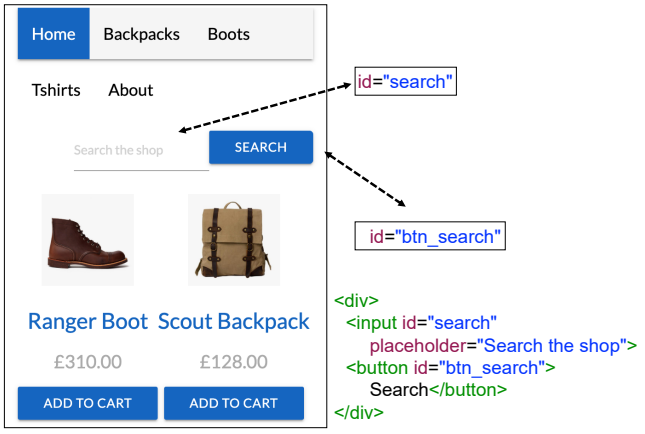
describes the definition, design, and the settings of the family of experiments. Section IV presents the quantitative and qualitative results of the experiments we conducted. Section V describes the related work, while conclusions and future work are given in Section VI.

II. The Page Object

The *Page Object* (PO) [4] pattern is a quite popular web test design pattern, which aims at improving the test case maintainability and reducing the duplication of code. A PO is a class that represents the web page elements as a series of objects, and that encapsulates the features of the web page into methods. Adopting the PO pattern in test cases allows testers to follow the *Separation of Concerns* design principle since the test scenario is decoupled from the implementation. Indeed, all the implementation details are moved into the POs, which represent bridges between web pages and test cases, with the latter only containing the test logic. Thus, all the functionalities to interact with a web page are offered in a single place, the PO, and can be easily called and reused within any test case. The use of the PO pattern reduces the coupling between web pages and test cases, promoting reusability, readability and maintainability of the test suites [5], [6].

Figure 1 shows a web page of ExpressCart, one of the web applications used in our experiment. The left-hand side of Figure 1 (top) shows a screenshot of the products page of ExpressCart. On the right-hand side, at the top, there is an excerpt of the source code that displays the search functionality in the products page. Both the input textbox and the search button have an `id` that identifies them uniquely in the hierarchical structure of the web page, called *DOM* (*Document Object Model*).

The bottom of Figure 1 shows a test case that exercises the search functionality of ExpressCart. The test case is implemented in Java, and it uses the *Selenium WebDriver* [1] framework to interact with the browser. In a nutshell, Selenium WebDriver interacts with the web browser, allowing a test case to navigate the web application under test as a user would (e.g., clicking on buttons and links, submitting forms, etc.). At line 2, the driver that will send commands to the Firefox browser is instantiated. The statement at line 3 starts up the Firefox browser and the statement at line 4 makes the browser load the first page of the web application under test (the products page of ExpressCart). At line 5, the driver is used to *locate* the input textbox of the search functionality within the web page through its `id` (i.e., `"search"`) and the `sendKeys` method of the resulting web element is used to write the value `"Boot"` into the input textbox. With the same location mechanism (i.e., the `id`) the statement at line 6 locates the search button and clicks on it so that the web page is updated with all the products whose name matches the search string. At line 7-8 the test locates the web element that contains the name of the first product in the resulting page through an `XPath`, a location technique that traverses the DOM tree starting from the root node to find the web element of interest. At line 9 the test checks that the name of the located product matches the string `"Ranger Boot"`, which is the product expected to be in the list of products after the search operation is performed. The last statement at line 10 closes the browser and ends the test case.




```
1  @Test      //test case without Page Objects
2  public void testSearchOK() {
3    WebDriver driver = new FirefoxDriver();
4    driver.get("https://www.expresscart.com");
5    driver.findElement(By.id("search")).sendKeys("Boot");
6    driver.findElement(By.id("btn_search")).click();
7    WebElement rangerBoot = driver.findElement(
8          By.xpath("(//h3[@class="product-title"])[1]"));
9    assertEquals("Ranger Boot",rangerBoot.getText());
10   driver.quit();
11 }
```

Fig. 1. ExpressCart products web page: excerpt of the page and the corresponding source code (top) and search test (bottom)

```
1  // ProductsPage Page Object
2  public class ProductsPage {
3    private final WebDriver driver;
4
5    public ProductsPage(WebDriver driver) {
6      this.driver = driver;
7    }
8    public ProductsPage search(String productName) {
9      driver.findElement(By.id("search")).sendKeys(productName);
10     driver.findElement(By.id("btn_search")).click();
11     return new ProductsPage(driver);
12   }
13   public String getProductName(int productOrder) {
14     return driver.findElement(By.xpath("(//h3[@class="product-title"])["
15         + productOrder + "]")).getText();
16   }
17 }
```

```
1  @Test            //test case with Page Objects
2  public void testSearchOK() {
3    WebDriver driver = new FirefoxDriver();
4    driver.get("https://www.expresscart.com");
5    ProductsPage productsPage = new ProductsPage(driver);
6    productsPage = productsPage.search("Boot");
7    assertEquals("Ranger Boot", productsPage.getProductName(1));
8    driver.quit();
9  }
```

Fig. 2. ExpressCart ProductsPage PO (top) and search test using PO (bottom)

Figure 2 shows how the same test case can be refactored using the PO pattern. The top part of the figure shows the code of the `ProductsPage` PO (a Java class) that models the products web page of EXPRESSCART (left hand side of Figure 1). In the figure, only two methods are shown, namely the `search` method and the `getProductName` method. The first one deals with filling in the input search textbox with the string passed as a parameter and clicking the search button; the second one retrieves the name of the product in the list, whose index is also passed as parameter. Therefore, the search test can be refactored as Figure 2 (bottom) shows in the bottom part. Assertions should not be part of a PO as including assertions mixes the responsibilities of providing access to page data with assertion logic, and leads to a bloated PO [4].

III. DEFINITION, DESIGN AND SETTINGS OF THE FAMILY

This section describes the definition, the design, and the settings of the family of experiments we conducted in a structured way, following the template and the guidelines by Wohlin *et al.* [7].

Table I summarizes the main elements of the experiments. The *goal* of the study is to investigate the impact of adopting the PO pattern on the development of web test cases (and thus of web test suites), with the purpose of evaluating how such pattern influences the development effort. The *quality focus* regards how the adoption of the PO pattern affects the time testers employ to develop web test cases.

The results of the experiments are interpreted regarding two *perspectives*: (1) researchers interested in empirically evaluating the effects of adopting the PO pattern and (2) quality managers who want to understand what the effort is, in terms of time, of writing the POs. This is the first step to understand whether the usage of POs pays off when facing actual web test case development tasks.

The *context* of the study is defined as follows: the participants are students who are assigned to web test case development tasks, while the objects are two open-source web applications. We collected empirical data from three experiments: the first with 9 MSc students of the University of Trento, attending a course on Security Testing; the second one involved 10 MSc students of the University of Genova, attending a course on Advanced Software Engineering; and finally, the last with 17 PhD students attending a PhD school on Automated Software Testing.

For replication purposes, the experimental package has been made available [8].

The research questions of the study are the following:

RQ_1: *What is the impact of adopting the PO pattern on the development effort of web test cases?*

To quantitatively investigate such a research question, we measured the *time* spent by each student to develop *correct* test cases in the assigned web test suite. We checked the correctness of the implemented test cases by executing them (we considered a test case as wrong if it fails) and by manually checking the presence of assertions.

In case of test suites implemented with the PO pattern, since it was challenging[1] to determine which POs methods were created during the development of the test cases, we applied an approximation. We decided to uniformly distribute the total time the student spent to develop the POs on all the correctly implemented test cases in that test suite. In other words, we charged each correctly implemented test case with a uniform fraction of the total POs development time.

RQ_2: *How does the impact of adopting the PO pattern change when it is distributed on a growing number of test cases?*

In order to answer RQ_2, we simulated the situation of larger test suites with respect to the ones assigned and implemented by the students in our experiments that are in the order of dozens of test cases. Then, similarly to RQ_1, we distributed the total time for the development of all the POs (if any) across all the correctly implemented test cases. In particular, we considered two situations. The first one in which the test suites are ten times larger, i.e., in the order of hundreds of

[1]Note that, when creating a test suite, the POs, and even the single PO methods can be refined during the development of different test cases. This makes it hard to assign the correct fraction of time spent on the modified portion of the PO to the correct test case. Another problem correlated to assigning the correct PO development time to each test case is connected with the tests development order: in fact, the development time is assigned to the first test case that requires a particular PO or PO method.

TABLE I
OVERVIEW OF THE FAMILY OF EXPERIMENTS

Goal	Investigate the impact of adopting the PO pattern on the development of a web test suite
Quality Focus	Effect of PO pattern adoption on time testers employ to develop web test cases
Context	Objects: EXPRESSCART and ADDRESSBOOK web applications Subjects: 19 (9+10) master students and 17 PhD students
Null Hypotheses	1. No difference in efficiency of development task PO-No/PO-Yes (case $1\times$) 2. No difference in efficiency of development task PO-No/PO-Yes, when test suite is $10\times$ larger (case $10\times$) 3. No difference in efficiency of development task PO-No/PO-Yes, when POs are already available (case INF)
Treatments	PO-Yes, PO-No
Co-Factors	Web Application (app), Degree (deg), Experiment (exp), Laboratory (lab), LoC of a test case (loc)
Dependent Variable	Time to correctly develop a test case

test cases (abbreviated with $10\times$ in comparison to $1\times$ used for RQ_1) than the original ones and the second one, where the size of the test suites is much bigger, such that the quote of the POs in terms of time on the single test case tends practically to zero (abbreviated with INF). In the second situation, we simulate the case in which the POs for a given web application under test are already available (e.g., from past development iterations or generated automatically by APOGEN [9]) or are used for the development of an extensive test suite.

Distributing the collected total time to develop all the POs in a web test suite across an increasing number of test cases (case $10\times$ and INF) is justified by the fact that the number of possible POs is limited by the number of web pages (e.g., login, home, settings, etc.) or components of web pages (navigation bar, menu, etc.). Instead, the number of test cases is only limited by the number of scenarios used to exercise a web application. Since the latter is usually much higher than the former, the cost of developing the POs on a single test case diminishes when the total number of test cases increases.

A. Main factor and treatments

Our experiment has one main factor (PO adoption) and two treatments: PO-Yes or PO-No. In the former case, the participant is asked to implement the test cases using the PO pattern (PO-Yes), while in the latter, the participant is asked to implement the test cases without adopting the PO pattern (PO-No).

B. Objects

The objects of the study are two web applications belonging to two different domains, namely EXPRESSCART [10] and ADDRESSBOOK [11]. These web applications have also been used in other empirical studies (e.g., ADDRESSBOOK is used by the authors of the SUBWEB [12], PESTO [13], and ROBULA+ [14] tools). We have already introduced EXPRESSCART in Section II, a full-fledged shopping cart built in Node.js, frameworks (Express) and non-relational DBMS (MongoDB). On the contrary, ADDRESSBOOK is a web-based address and phone book, contact manager and organizer developed with more standard technologies (PHP as server-side language and MySQL as relational DBMS). Both the applications used in our family of experiments are simple to use and straightforward to understand, although non-trivial. ADDRESSBOOK is developed without any server-side framework and it features $30k$ PHP LoC and $1.3k$ JavaScript LoC, excluding libraries. EXPRESSCART is built with the Express framework (a JavaScript server-side framework) and it features $3k$ JavaScript LoC, excluding libraries. The dimension of EXPRESSCART could appear small, but it is in line with the size of web applications built using frameworks. Indeed, Ocariza et al. [15] report an average of 1689 LoC for a dataset of web applications developed with the AngularJS framework.

C. Participants

The experiments were conducted in research laboratories under controlled conditions. The participants were 19 MSc in total and 17 PhD Computer Science students. In the case of MSc students, E2E testing, Selenium and POs were explained during the course in which the experiment was conducted. In the case of PhD students, the same concepts given to MSc students were introduced in a practical lesson of the PhD school, prior to the experiment. We used the same slides for all the explanations. Before the experiment, during a training session, all participants (both MSc and PhD) implemented a training web test suite for a web application different from the two used in the experiments (i.e., PETCLINIC [16]). Each participant had to implement the web test suite with and without the PO pattern, similarly to the tasks executed during the experiments.

D. Dependent Variable and Hypothesis Formulation

Our experiment has only one dependent variable, which is the time to develop a test case. We used the time to correctly develop a test case as a measure of *efficiency* for the test case development task. We took into account the adoption of the PO pattern, as previously explained in Section III.

In order to keep track of the time each participant spent on each test case (and PO, in case of PO-Yes treatment) of the given test suite, we used *Rabbit* [17], a statistics tracking plug-in for Eclipse. Rabbit runs in background to record how the user spends her time within Eclipse. It is then possible to export such statistics in XML format. Since, among other statistics, Rabbit tracks the time spent by the user working on Java elements such as classes, we organized each test case in a different Java class to measure the test case development time. Another useful feature of Rabbit is that it only tracks the time when Eclipse is active, meaning that if Eclipse's window is not focused, tracking is paused.

Based on the study definition reported above, we formulated the following null hypotheses to be tested:

H_{01}: *Developing a correct test case adopting the PO pattern is as efficient as developing a correct test case without the PO pattern.*

H_{02}: *Developing a correct test case adopting the PO pattern is as efficient as developing a correct test case without the PO pattern, when the cost for developing all POs is distributed across a test suite ten times larger than the actual one.*

H_{03}: *Developing a correct test case adopting the PO pattern is as efficient as developing a correct test case without the PO pattern, when POs are already available or their development cost is negligible because it is distributed across an extensive test suite.*

Since it is difficult to foresee a possible trend, we opted for the most conservative choice and formulated the null hypotheses as a *two-tailed* hypotheses. In fact, on the one hand, developing test cases without POs could be faster since there is no overhead associated with the creation of POs. On the other hand, using the POs could make development faster, since the overhead for creating the POs might be balanced by their reuse across test cases.

TABLE II
EXPERIMENTAL DESIGN (PO-YES = USING THE PO, PO-NO = WITHOUT USING THE PO)

	Group A	Group B	Group C	Group D
Lab 1	ADDRESSBOOK PO-Yes	ADDRESSBOOK PO-No	EXPRESSCART PO-No	EXPRESSCART PO-Yes
Lab 2	EXPRESSCART PO-No	EXPRESSCART PO-Yes	ADDRESSBOOK PO-Yes	ADDRESSBOOK PO-No

E. Co-factors

We also considered whether the efficiency is influenced by other factors and how such factors interact with the main factor. In particular, we considered the following co-factors:

1) The degree of the participants (MSc or PhD);
2) The object (i.e., the web application). Since we adopted a balanced design with two objects, participants could exhibit different performance on different objects. Hence the object is also a factor;
3) The experiment session (i.e., lab). We measured whether any learning effect occurred between the two labs;
4) The LoC of each test case (i.e., an approximate proxy for the complexity of a test case). We measured whether the impact of the PO pattern varies when the test case size (i.e., the number of statements) varies.

F. Experiment Design

The experiment adopts a counter-balanced design to fit two lab sessions (see Table II). Participants were split into four groups balancing as much as possible their expertise in software testing, test automation and programming. In the case of MSc students, the division into groups was done considering the evaluation of previous mandatory exercises executed during the software engineering courses. In the case of PhD students, the expertise was inferred by asking participants their skills about (1) Java language, (2) usage of the Eclipse IDE, (3) software testing and test case automation, and (4) DOM locators.

The counter-balanced design adopted in our family of experiments ensures that each participant works on different objects in the two labs, receiving each time a different treatment, which is the best choice when a limited number of participants is available. Moreover, a counter-balanced design limits as much as possible learning effects [18].

G. Material, Procedure and Execution

The experiments took place in laboratory rooms and the participants participated in two laboratory sessions (Lab 1 and Lab 2) on two different days.

Before the first laboratory, during the training session, we gave to the participants instructions on how to set up their laptops, to avoid wasting time in software configuration during the actual experimental session. Moreover, we distributed the following material:

- links to the software needed to perform the experimental tasks: the Eclipse IDE, Docker, Firefox, ChroPath [19] (a Firefox add-on to create DOM locators) and Rabbit [17];
- links useful to download the two web applications used as objects. ADDRESSBOOK and EXPRESSCART were installed in a Docker container for which we created a Docker image. Participants only had to download the Docker image and start a Docker container using that image with the command-line instructions we provided;

The day of the first laboratory we gave to the participants a link useful to download:

1) a PDF file containing the detailed, step-by-step, textual description of the test cases they had to implement for the assigned web application. The test case textual descriptions have been defined by the Authors, during the planning of the experiment, to cover the most relevant functionalities of the two web applications (i.e., the objects).
2) a Java project containing the skeleton of the test suite to be implemented (precisely, containing 20 test skeletons for ADDRESSBOOK and 15 for EXPRESSCART), where the Selenium test cases (one test case per class) were left unimplemented.

While we specified the test case skeletons for each web application, we did not specify anything about the POs to be implemented. The reason is that the choices regarding which POs to create and how to create them for the given test scenarios are modelling choices associated with the adoption of the PO pattern and we did not want to introduce any bias on such choices.

Moreover, on the same day of the first laboratory, we gave to participants a description of the experimental procedure, but no reference was made to the study hypotheses. The experiment has been carried out according to the following instructions:

1) Import the Java project of the assigned web application containing the test case skeletons into Eclipse;
2) Activate Rabbit plug-in in Eclipse;
3) Develop the test cases, one after the other, following the given descriptions;
4) Create an archive containing the modified test suite project and send it to the experimenter by email;

During the experiment, teaching assistants were present in the laboratory to prevent collaboration among participants and to verify that the experimental procedure was respected.

H. Post-experiment questionnaire

After the two lab sessions, at the end of the second day, participants filled a post-experiment survey questionnaire. This is useful for receiving opinions on the experiment and on the PO pattern and for finding justifications for the quantitative observations. Answers were given on the following five-item Likert scale [20]: strongly agree, agree, neutral, disagree, and strongly disagree. The questionnaire the participants filled in consists of five Likert scale-based questions (we

asked participants to choose a Likert-level for evaluating each provided statement):

Q_1 : *Developing a PO based test suite requires more time than developing a test suite without POs*

Q_2 : *Developing a PO based test suite is more difficult than developing a test suite without POs*

Q_3 : *The effort required to write a test case together with a PO, compared to writing a test case with no PO, was substantially higher*

Q_4 : *The additional effort required to write a PO is worth being spent, because then writing test cases is substantially easier*

Q_5 : *The additional effort required to write a PO is worth being spent, because then test cases are of higher quality*

I. Pilot experiment

To assess the experimental material and to get an estimate of the time needed to accomplish the tasks, a pilot experiment with two MSc students in Computer Science at University of Genova was performed. Both students had a consolidated experience in developing E2E test cases with and without the PO pattern, hence those concepts were not explained to them before the actual development tasks. The students finished the development of the assigned test suites (each student with both treatments) in about three hours and gave us feedback on how to improve the experimental material, in particular concerning the description of the test cases to develop. Based on the pilot experiment and taking into account the time constraints of the labs in which the real experiments had to be carried out, we decided to give participants two hours per laboratory (i.e., four hours in total).

J. Analysis Procedure

To analyse the results of the experiment, we used *Generalized Linear Models* [21] (*GLMs* for short). GLM incorporates a number of different statistical models: *ANOVA*, *ANCOVA*, *MANOVA*, *MANCOVA*, ordinary linear regression, t-test and F-test. To test the efficiency of participants in performing the test development tasks (H_{01}, H_{02}, H_{03}) we used a generalized linear model (with family = gaussian) fitting the dependent output variable efficiency measured in terms of time as a function of the independent input variables (all factors, including the main factor, i.e., PO adoption). A generalized linear model allows us to test the statistical significance of the influence of the independent factors on the time required to complete the test case development tasks. We assume as usual significance at 95% confidence level (α=0.05), so we reject the null-hypotheses having p-value < 0.05. Moreover, in order to understand the magnitude of the statistical difference, we computed the effect size. In particular, we computed the Cohen's d [22]. The effect size is considered negligible if $|d| < 0.2$; small if $0.2 \leq |d| < 0.5$; medium if $0.5 \leq |d| < 0.8$; large if $|d| \geq 0.8$.

IV. Results

This section reports and discusses the results for each RQ.

A. RQ_1: Impact of adopting the PO pattern

The boxplots reported in Figure 3 and Figure 4 show the time required for correctly developing the test cases for EXPRESSCART and ADDRESSBOOK, divided by treatment. The sum of all the possible test cases developed by the participants is 1260 (the 19 MSc and 17 PhD participants developed up to 20 tests for ADDRESSBOOK and 15 for EXPRESSCART). Since we consider only correctly implemented test cases, boxplots show only the 633 correct test cases, split into 370 test cases developed without using the PO pattern and 263 with the PO pattern. From the boxplots, it is apparent that participants adopting the PO pattern were less efficient, i.e., employed substantially more time for developing the test cases. This trend is present in both applications, even if for EXPRESSCART the difference between the two treatments is more evident.

Table III reports the analysis of the experimental data using GLM. The model takes into account not only the effect of the main factor (*po*, with or without PO pattern adoption) but also all the co-factors we considered in our experimental setting,

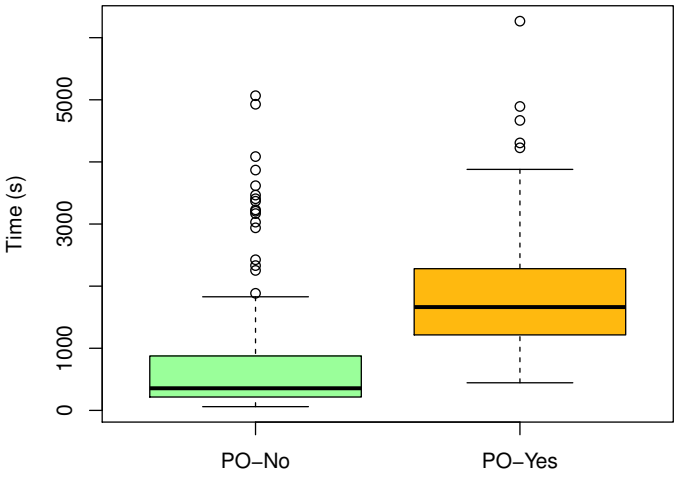


Fig. 3. EXPRESSCART: Time required for developing each Test Case partitioned by PO adoption

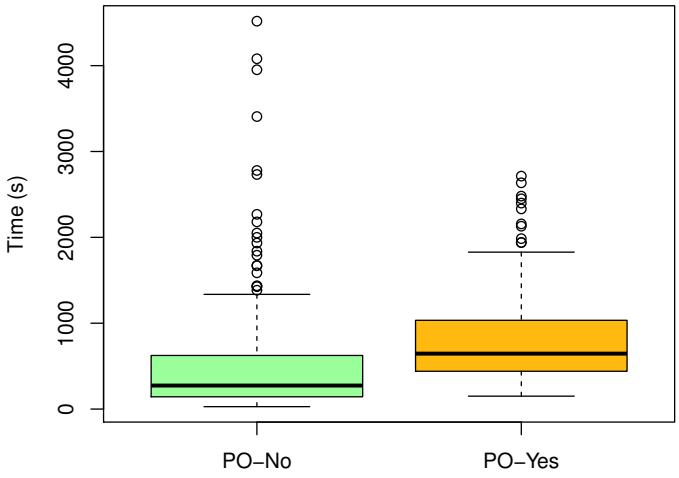


Fig. 4. ADDRESSBOOK: Time required for developing each Test Case partitioned by PO adoption

TABLE III
GENERALIZED LINEAR MODEL (GLM) ANALYSIS OF TIME WITH RESPECT TO THE *po* MAIN FACTOR AND THE OTHER CO-FACTORS

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-868.4456	284.5748	-3.05	0.0024
po	614.3154	79.4645	7.73	**0.0000**
app	669.4192	88.7260	7.54	**0.0000**
deg	471.8295	188.8062	2.50	**0.0128**
exp	-244.1798	116.5279	-2.10	**0.0366**
lab	-245.0361	84.1699	-2.91	**0.0038**
loc	7.2324	2.9231	2.47	**0.0137**

i.e., the application (*app*, EXPRESSCART or ADDRESSBOOK), the degree (*deg*, either MSc or PhD), the experiment (*exp*, first, second or third replication), the laboratory (*lab*, either first or second lab), and the length of the test cases (*loc*, lines of code). Statistically significant cases are in boldface, considering a 95% confidence level ($\alpha = 0.05$).

Data in Table III show that all the considered factors achieved a statistical significance. Focusing on the main factor, *po*, and looking at the positive value of the *Estimate* column, we can conclude that adopting the PO pattern (i.e., moving from the PO-No treatment to the PO-Yes one) increases significantly the time required to develop the test cases.

For this reason, we can reject H_{01} and conclude that *Developers adopting the PO pattern are **less efficient** than developers not adopting it*.

This result is reasonable, since adopting the PO pattern brings additional work compared to the development of bare test cases and savings occur only when POs are reused many times. Indeed, even if the steps of the test cases are the same, PO adoption implies: (1) the design of the PO navigation model, i.e., an abstract view of the navigational structure of the web application, where each node corresponds to a PO; (2) the implementation of the POs, including DOM locators and navigation/observational methods. This conceptual (PO modelling) and practical effort impacts negatively on the development time when the total number of developed test cases is low, e.g., 10-20, as in our experiment. On the contrary, we conjecture that when the number of test cases increases, the reuse of PO methods in different test cases could reverse the trend and bring improvements in terms of development efficiency. This conjecture is investigated in RQ2.

From Table III we can also analyse the role of the other factors influencing the dependent variable. For the most influential ones (*app* and *lab*) we also report the interaction plots with the main factor *po*. Interaction plots visually represent the interaction between the effects of two factors.

Thus, let us first consider the application (*app*) and the laboratory (*lab*). We can notice that both of them have a significant effect. From the interaction plot reported in Figure 5, we can notice that the participants spent more time in developing test cases for EXPRESSCART. Moreover, the difference in time is amplified when POs are adopted (PO-Yes). The reason may be that EXPRESSCART is a single-page

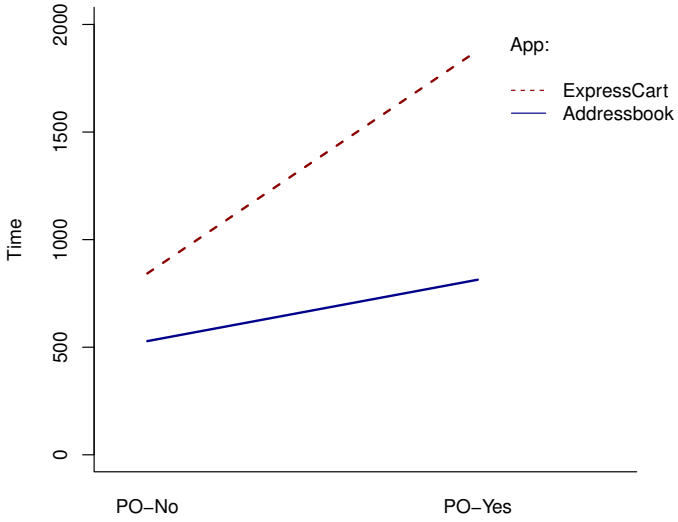


Fig. 5. Interaction plot of Time between Treatment (PO Pattern YES, NO) & Application (EXPRESSCART, ADDRESSBOOK); diverging/converging lines indicate potential interactions.

web application, while ADDRESSBOOK is a multi-page web application. The specific nature of single-page applications complicates the design/implementation of the POs. In fact, in multi-page web applications, the creation of a PO can be guided by the URL of the web page (since each web page is identified by a unique URL) whereas, in single-page web applications, the URL could be no longer meaningful (as in the case of EXPRESSCART) and each PO has to be created based on the logical meaning of the displayed web page. The difference in time in the *PO-No* case can be explained considering that EXPRESSCART is slightly more complex than ADDRESSBOOK.

From the interaction plot reported in Figure 6, it is quite clear that adopting the PO pattern in the first lab required more time than adopting the pattern in the second lab. This is probably

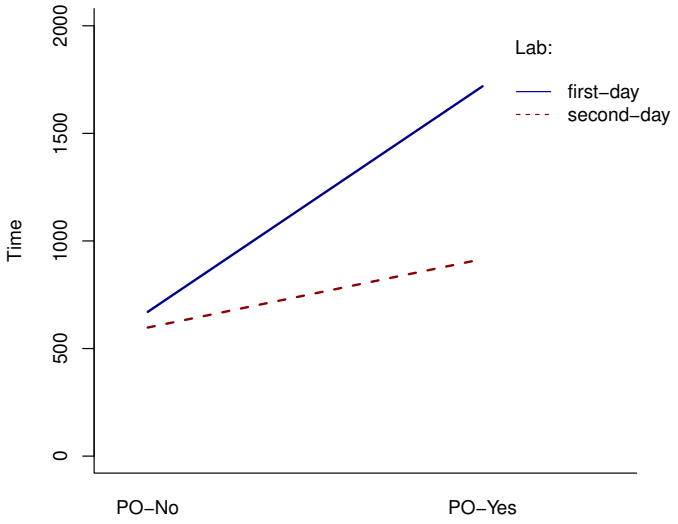


Fig. 6. Interaction plot of Time between Treatment (PO Pattern YES, NO) & Laboratory (First Day, Second Day); diverging/converging lines indicate potential interactions.

due to a lack of training and experience. We provided some training on the PO pattern, but it was probably not enough: the PO design pattern needs time and practice to be applied effectively and efficiently. From this result, we can speculate that the effect on the time required to develop a test case with the PO pattern is amplified in our experiment w.r.t. the one that is likely to be observed with skilled developers (i.e., professional testers having experience in E2E testing).

Looking at the co-factor *deg*, we can observe that the degree of the participants (i.e., being an MSc or PhD student) had a significant impact in the expected direction. This is reasonable since usually, the skills of PhDs attending a doctoral school on testing are higher than the skills of MSc students. For the same reason, the *exp* co-factor is also significant. Not surprisingly, the *loc* co-factor is significant: longer test cases require significantly more time to be developed than shorter test cases.

Summary: the adoption of the PO pattern decreases the efficiency of testers developing web test cases. This is true in particular when: a) small test suites are developed (test suites comprising 10-20 test cases), due to a low reuse of PO methods across test cases; and, b) in the presence of testers poorly skilled in E2E testing. Also the type of application (single vs. multi-page) plays an important role: single page applications complicate PO design/implementation, increasing the gap between poorly and very skilled developers, in terms of efficiency in PO development.

B. RQ_2: Test Suite Size Influence

Table IV reports the GLM analysis in the simulated case in which the cost of developing all POs is distributed across a test suite ten times larger (i.e., case $10\times$) than the real test suites developed in the experiments.

Also in this case, as done for $\mathbf{RQ_1}$, the model takes into account all the considered co-factors. Statistically significant cases are in boldface.

TABLE IV
GENERALIZED LINEAR MODEL (GLM) ANALYSIS OF TIME (CASE $10\times$) WITH RESPECT TO THE *po* MAIN FACTOR AND THE OTHER CO-FACTORS

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	82.6199	254.9939	0.32	0.7461
po	-106.2981	71.2044	-1.49	0.1361
app	435.1771	79.5032	5.47	**0.0000**
deg	278.6427	169.1802	1.65	0.1002
exp	-166.5301	104.4151	-1.59	0.1114
lab	-133.5374	75.4206	-1.77	0.0773
loc	11.2036	2.6193	4.28	**0.0000**

Table V reports the GLM analysis in the simulated case in which the POs are already available, are automatically generated or are reused a very large number of times, which corresponds to the case where the time needed for developing the POs approaches zero (i.e., the time required to develop page objects is negligible compared to the time required to develop test cases).

TABLE V
GENERALIZED LINEAR MODEL (GLM) ANALYSIS OF TIME (CASE INF) WITH RESPECT TO THE *po* MAIN FACTOR AND THE OTHER CO-FACTORS

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	188.2938	253.7993	0.74	0.4585
po	-186.3663	70.8708	-2.63	**0.0088**
app	409.1502	79.1307	5.17	**0.0000**
deg	257.1776	168.3876	1.53	0.1273
exp	-157.9024	103.9259	-1.52	0.1293
lab	-121.1487	75.0673	-1.61	0.1072
loc	11.6448	2.6070	4.47	**0.0000**

Focusing on the main factor (*po*), the two tables show that when the number of test cases increases and PO methods are reused more, the advantage of adopting the PO pattern becomes apparent. In fact, comparing the main factor in the three tables, we can see that: for $1\times$ (Table III) the estimate column is positive and the contribution significant, for $10\times$ (Table IV) the estimate column becomes negative, meaning that now the usage of the PO pattern diminishes the time to develop test cases instead of increasing it, but the contribution is not significant (p-value = 0.1361) and finally, for INF (Table V) the estimate column remains negative but the contribution turns to be significant.

As a consequence, we cannot reject H_{02} but we can reject H_{03}. Thus, we can conclude that: *Developers adopting the PO pattern are* ***more efficient*** *than developers not adopting it only when POs are already available or their development cost is distributed across a reasonably large test suite (i.e., one covering the web application functionalities in depth, with multiple test cases).*

These results are also confirmed by looking at Table VI showing the effect size (Cohen's d) for the cases $1\times$, $10\times$ and INF computed between the two distributions of the times (PO-Yes vs. PO-No). To appreciate the effect of the application factor, results are partitioned by application.

TABLE VI
COHEN'S D OF TIME VS PO ADOPTION (PARTITIONED FOR APPLICATION) IN CASES $1\times$, $10\times$ AND INF

	$1\times$ case	$10\times$ case	INF case
ADDRESSBOOK	-0.459126	0.243672	0.326870
EXPRESSCART	-0.939566	0.031810	0.150949

In case $1\times$ the Cohen's d values are negative for both web applications. This means that adopting the PO pattern has a (large for EXPRESSCART and medium for ADDRESSBOOK) negative impact on the time required to develop the test cases (i.e., it increases the time). On the contrary, in the $10\times$ case, both Cohen's d values become positive (i.e., POs diminish development time). In particular, for ADDRESSBOOK we have a "small" positive effect, while for EXPRESSCART the effect is "negligible", but still positive. Finally, the same trend is appreciable also in the INF case where the effect size values

still increase for both applications, but by a much greater amount.

Concerning the co-factors, the significant ones in the simulated cases are *loc* and *app* (see Table IV and Table V). This means that even when the PO development cost is distributed across larger test suites, the contribution of these co-factors continues to be significant.

Summary: our results show that the effect of adopting the PO pattern on the time required to develop the test cases is strongly dependent on the reuse level of the PO methods.

When the test cases reuse few methods in the POs, as in case $1\times$, the total development time of the test suite is higher when the PO pattern is adopted. On the contrary, when each method is reused at least ten times (as in case $10\times$) the adoption of the PO pattern becomes convenient. It is important to notice that reuse across hundreds of tests in a typical industrial context is not difficult to reach.

C. Post Experiment Questionnaire

We used the answers to the questions Q_1 to Q_5 of the post-experiment questionnaire to gain insights on the participants' activity. Results are summarized in Table VII, split by MSc and PhD students and considered all together. All students agree in their answers to questions Q_2, Q_4 and Q_5. In particular, both categories of students are neutral on the difficulty of PO adoption (Q_2), they think that using the PO pattern eases the development of web test cases (Q_4) and improves web test cases quality (Q_5), i.e. test cases are more maintainable and readable when the PO pattern is adopted.

TABLE VII
ANALYSIS OF POST QUESTIONS Q_1-Q_5 FOR EACH DEGREE LEVEL AND AGGREGATED. LIKERT SCALE LEVELS REPRESENT THE MEDIAN FOR EACH QUESTION

	MSc (Median)	PhD (Median)	All (Median)
Q_1: PO-Yes requires more time	Agree	Disagree	Agree
Q_2: PO-Yes is more difficult	Neutral	Neutral	Neutral
Q_3: PO-Yes requires more effort	Agree	Neutral	Agree
Q_4: PO-Yes test case writing easier	Agree	Agree	Agree
Q_5: PO-Yes test cases higher quality	Agree	Agree	Agree

Concerning Q_3, there is almost agreement. Overall, the perceived effort in terms of time is in line with the quantitative results shown in Table III. On the contrary, there is a remarkable difference for Q_1. For PhD students, the usage of POs does not require more time w.r.t. developing a bare test suite, while MSc students perceive the PO adoption more expensive in terms of time.

We tried to understand the reason for this difference by analysing the fraction of the test suites implemented during the experiments. We found that MSc students correctly implemented about 30% of the test cases with POs and 50% without POs. On the contrary, PhD students implemented about 55% and 65% of the test suites respectively in the two cases. Thus, the difference between using or not the POs is clearly appreciable in the case of the MSc (close to a $2\times$ factor) while it is less evident in the case of PhD students. This could have distorted the perceptions of PhD students.

D. Threats to validity

This section discusses the threats to validity that could affect our results: *internal*, *construct*, *conclusion* and *external* validity threats [7].

Internal validity threats concern factors that may affect a dependent variable, in our case the time required to develop the test cases. Since students participated in two labs, a learning and fatigue effect might have intervened. Students were previously trained and the chosen experimental setting should have limited learning effects. However, our analysis of co-factors shows some learning effect, in particular when the PO pattern is adopted (see Figure 6). This shows that mastering the PO pattern is not trivial and requires substantial skills and experience. To reduce any fatigue effect, we chose to execute the two lab sessions on different days. An additional threat, that could add an additional confounding factor to the experiment, concerns the specific applications used for the experiment belonging to two different categories (single vs multi-page web apps). Another possible threat could be related to the textual descriptions of the test case, which affect the test cases and the page objects to be implemented, and which could give an advantage to a treatment with respect to the other. We mitigated this threat by defining the test cases as usually done in functional testing, i.e., by covering the most relevant functionalities of the web application under test, regardless of the POs that are eventually required for their implementation.

Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measured the capability of a participant to develop a correct test case and the time needed to do it. Thus, this threat is related to how the correctness of the test case and time were measured. The measurements we conceived to assess the correctness of test cases are as objective as possible. Even if we have no guarantee that the implemented test cases considered in the analysis are correct, executing them and checking the presence of assertions gave us a good level of confidence. The time needed to develop the test cases and the POs was recorded by an automated tool (Rabbit), reducing as much as possible inaccuracies.

Threats to *conclusion validity* concern issues that may affect the ability to draw a correct conclusion. They can be due to the sample size of the family of experiments (36 participants in total) that may limit the capability of statistical tests to reveal any effect and to the chosen statistical tests. In our family of experiments, we chose to use GLM (as done in other recent empirical Software Engineering papers, e.g., [23], [24]) to develop an optimal predictive model to predict the time required to develop the test cases because this statistical test is particularly robust (i.e., it does not give false rejections

of the null hypothesis) under deviations from normality and homoscedasticity. Post experiment questionnaires, mainly intended to get qualitative insights, were designed using standard structure and scales [20].

Threats to *external validity* can be related to: (i) the choice of simple web applications as objects and (ii) the use of students as experimental participants. The size and complexity of the two web applications are suitable for the time allotted to the experiment but are significantly smaller than that of most industrial web applications. It is possible that some phenomena do appear only as the size scales up (both the size of the web application and of the corresponding test suite) and, thus, they are not observable in our experiments. We have simulated larger sizes of test suites by distributing the PO development cost across a larger number of test cases. As far as the participants are concerned, we are aware that the expertise of students may be different from that of professionals. However, finding professionals available to conduct a demanding experiment as the one we designed is not easy. This threat was mitigated by: (a) considering students with different levels of education (MSc and PhD); and, (b) performing a co-factor analysis by *deg*, so as to take into account different skills and experience, as occurring in our two populations of MSc and PhD students.

V. Related Work

Empirical evidence on test automation is well represented in the literature [25]–[27], while specific studies aimed at assessing the effectiveness and usefulness of page objects as a way to abstract the implementation details of a GUI are extremely rare.

Martin Fowler was the first to describe this pattern under the name Window Driver [1]. However, the term Page Object has been popularised by the Selenium web testing framework, becoming the standard, de-facto name. Van Deursen [28] goes one step further and proposes a state-based formalization of page objects, which can help testers to build the PO model, representing both page objects and how to navigate from a page object to another.

The empirical study by Leotta et al. [5] shows that test suites developed with a programmable approach (Selenium WebDriver) involve higher development but lower maintenance effort w.r.t. test suites developed with a capture-and-replay approach (Selenium IDE). This result is also due to the introduction of the PO design pattern, able to decouple the test logics from that of the application. The goal of our study is different, even if related to the Leotta et al.'s study. In fact, we are interested in quantifying the cost of using the PO pattern within the programmable approach, rather than comparing two different test case development approaches. In particular, our goal was to understand if there is a test suite size at which the benefits of reusing POs across test cases compensate for the extra development effort required.

Another study, conducted in an industrial setting [29], indicates a strong reduction in terms of time required and the number of modified LoC to maintain a test suite when the PO pattern is used. While in our study we evaluated only the potential cost of introducing POs, considering also the benefits of POs during maintenance activities is another interesting direction that we will consider in future controlled experiments.

Building POs for web applications is considered by practitioners as an expensive task, which is usually performed manually. For this reason, some researchers proposed a solution, implemented in the tool APOGEN [9], able to provide a considerable degree of automation, hence reducing the effort for the creation of POs. Although our goal is completely different, both works are motivated by the adoption effort involved in the usage of POs in test cases.

Yu et al. [30] claim to have developed a prototype capable of automatically generating POs and PO based test suites. Their prototype is based on feedback directed random test generation. The SUBWEB tool [31] uses the PO model defined by developers and applies a search-based approach, instead of a random one, to generate test inputs and feasible navigation paths.

While the perspective of automatically generating the POs is very attractive and potentially very useful for practitioners, existing proposals [9], [30] are still research prototypes and their output requires manual intervention to fix the reverse-engineered abstractions. On the contrary, our empirical study shows that manual PO development is affordable and pays off if the test suite size is reasonably large, at least ten times the size of a test suite that a student can develop in two hours.

VI. Conclusions and Future Work

We have conducted a family of three controlled experiments with MSc and PhD students to measure how the adoption of the PO design pattern impacts the time to develop a web test suite of small size (10 to 20 test cases). We then simulated the development of larger test suites by distributing the PO development cost across a larger number of tests. The results of our statistical analyses indicate that the benefits of PO adoption in terms of higher developer's efficiency can be appreciated only with reasonably large test suites, estimated as at least $10\times$ larger than those developed by the students involved in our two hours sessions. In a typical industrial web development context, it is quite common to create test suites with hundreds of test cases, hitting the point at which the PO pattern pays off. It should also be noticed that the developers' skills play a statistically significant role. In an industrial context, where web testing skills and domain knowledge are higher than those reached by students during our two hours training session, the benefits of PO adoption are amplified and are likely to become apparent even with smaller size increase than our estimated $10\times$ factor. Hence, our results can be deemed as conservative overestimates of the turning point at which POs affect the developers' efficiency in a positive way.

In our future work, we will design and conduct controlled experiments to measure the effects of the PO pattern during software and test suite evolution. Similarly to Leotta et al. [29], we also intend to conduct industrial case studies on the adoption of the PO pattern.

REFERENCES

[1] "Selenium WebDriver," https://www.seleniumhq.org/projects/webdriver/.

[2] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] F. Ricca, M. Leotta, and A. Stocco, "Three open problems in the context of E2E web testing and a vision: NEONATE," ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2019, vol. 113, pp. 89–133. [Online]. Available: https://doi.org/10.1016/bs.adcom.2018.10.005

[4] M. Fowler, "PageObject," http://martinfowler.com/bliki/PageObject.html.

[5] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," in *Proceedings of 20th Working Conference on Reverse Engineering*, ser. WCRE 2013. IEEE, 2013, pp. 272–281. [Online]. Available: https://doi.org/10.1109/WCRE.2013.6671302

[6] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Approaches and tools for automated End-to-End web testing," *Advances in Computers*, vol. 101, pp. 193–237, 2016. [Online]. Available: https://doi.org/10.1016/bs.adcom.2015.11.007

[7] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000. [Online]. Available: https://doi.org/10.1007/978-1-4615-4625-2

[8] "Complete experimental package for replication purposes," http://sepl.dibris.unige.it/ICST2020-exp-PO-replication.php.

[9] A. Stocco, M. Leotta, F. Ricca, and P. Tonella, "APOGEN: Automatic page object generator for web testing," *Software Quality Journal (SQJ)*, vol. 25, no. 3, pp. 1007–1039, 2017. [Online]. Available: https://doi.org/10.1007/s11219-016-9331-9

[10] "ExpressCart: A fully functioning Node.js shopping cart with Stripe, PayPal and Authorize.net payments." https://github.com/mrvautin/expressCart, 2019.

[11] "PHP Address Book: simple, web-based address and phone book," https://sourceforge.net/projects/php-addressbook, 2017.

[12] M. Biagiola, A. Stocco, A. Mesbah, F. Ricca, and P. Tonella, "Web test dependency detection," in *Proceedings of 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019, 2019, pp. 154–164. [Online]. Available: https://doi.org/10.1145/3338906.3338948

[13] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "PESTO: Automated migration of DOM-based web tests towards the visual approach," *Journal of Software: Testing, Verification and Reliability (STVR)*, vol. 28, no. 4, p. e1665, 2018. [Online]. Available: https://doi.org/10.1002/stvr.1665

[14] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "ROBULA+: An algorithm for generating robust XPath locators for web testing," *Journal of Software: Evolution and Process (JSEP)*, vol. 28, no. 3, pp. 177–204, 2016. [Online]. Available: https://doi.org/10.1002/smr.1771

[15] F. S. Ocariza Jr, K. Pattabiraman, and A. Mesbah, "Detecting unknown inconsistencies in web applications," in *Proceedings of 32nd International Conference on Automated Software Engineering*, ser. ASE 2017. IEEE, 2017, pp. 566–577. [Online]. Available: https://doi.org/10.1109/ASE.2017.8115667

[16] "AngularJS and Spring Boot version of the Spring Petclinic sample application," https://github.com/spring-petclinic/spring-petclinic-angularjs, 2019.

[17] "Rabbit: a statistics tracking plug-in for Eclipse." https://code.google.com/archive/p/rabbit-eclipse/, 2011.

[18] H. Motulsky, *Intuitive biostatistics: a non-mathematical guide to statistical thinking*. Oxford University Press, 2010.

[19] "AutonomIQ offers developers a platform (ChroPath) to generate and validate unique selectors like in the devtools panel," https://addons.mozilla.org/en-US/firefox/addon/chropath-for-firefox/, 2019.

[20] A. N. Oppenheim, *Questionnaire design, interviewing and attitude measurement*. Bloomsbury Publishing, 2000.

[21] J. A. Nelder and R. W. M. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society. Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972. [Online]. Available: https://doi.org/10.2307/2344614

[22] J. Cohen, "A power primer," *Psychological Bulletin*, 1992. [Online]. Available: https://doi.org/10.1037/0033-2909.112.1.155

[23] M. Leotta, M. Cerioli, D. Olianas, and F. Ricca, "Two experiments for evaluating the impact of Hamcrest and AssertJ on assertion development," *Software Quality Journal (SQJ)*, 2020. [Online]. Available: https://doi.org/10.1007/s11219-020-09507-0

[24] M. Ceccato and R. Scandariato, "Static analysis and penetration testing from the perspective of maintenance teams," in *Proceedings of 10th International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM 2016. ACM, 2016, pp. 25:1–25:6. [Online]. Available: https://doi.org/10.1145/2961111.2962611

[25] F. Dobslaw, R. Feldt, D. Michaelsson, P. Haar, F. G. de Oliveira Neto, and R. Torkar, "Estimating return on investment for GUI test automation tools," *CoRR*, vol. abs/1907.03475, 2019. [Online]. Available: http://arxiv.org/abs/1907.03475

[26] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on visual GUI testing," *Information and Software Technology*, vol. 73, pp. 66–80, 2016. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.01.012

[27] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: Empirical observations," *Advances in Software Engineering*, vol. 2010, no. Article 4, 2010. [Online]. Available: https://doi.org/10.1155/2010/620836

[28] A. van Deursen, "Testing web applications with state objects," *Communications of ACM*, vol. 58, no. 8, pp. 36–43, 2015. [Online]. Available: https://doi.org/10.1145/2755501

[29] M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro., "Improving test suites maintainability with the Page Object pattern: an industrial case study," in *Proceedings of 6th International Conference on Software Testing, Verification and Validation Workshops*, ser. ICSTW 2013. IEEE, 2013, pp. 108–113. [Online]. Available: https://doi.org/10.1109/ICSTW.2013.19

[30] B. Yu, L. Ma, and C. Zhang, "Incremental web application testing using page object," in *Proceedings of 3rd Workshop on Hot Topics in Web Systems and Technologies*, ser. HOTWEB 2015. IEEE, 2015, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/HotWeb.2015.14

[31] M. Biagiola, F. Ricca, and P. Tonella, "Search based path and input data generation for web application testing," in *Proceedings of 9th International Symposium on Search Based Software Engineering*, ser. SSBSE 2017, 08 2017, pp. 18–32. [Online]. Available: https://doi.org/10.1007/978-3-319-66299-2_2