University College of Enterprise and Administration in Lublin

Faculty of Technical Sciences

Major: **Computer Science**

AUTOMATED TESTING OF WEB APPLICATIONS IN JAVA AND IN RUBY. A CASE STUDY

AUTOMATYZOWANIE TESTÓW APLIKACJI WEBOWYCH W JAVIE I W RUBYM. STUDIUM PRZYPADKU

Author: Michał Rakowski

Student ID: 17063

Academic advisor: Karol Kuczyński, Ph.D.

Lublin 2019

Oświadczenie kierującego pracą.

Oświadczam, iż niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

…… ……..

(data) (podpis)

Oświadczenie autora pracy.

Świadomy odpowiedzialności prawnej oświadczam, iż niniejsza praca dyplomowa została napisana przeze mnie samodzielnie nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. Nr 24, poz 83 z poźn. zm.) oraz dóbr osobistych chronionych prawem cywilnym dodatkowo nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami. Oświadczam również, że przedstawiona praca nie była wcześniej podmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni. Ponadto oświadczam, iż niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

....

(data) (podpis)

CONTENTS

GLOSSARY

SUT – System Under Test, the application/program which is tested

E2E – end-to-end tests, tests which involved the whole system, usually accessed through the GUI

GUI – graphical user interface, what the user sees, the text fields, buttons, dropdown lists user interacts with

BAT – Black-box Automated Tests, E2E, GUI-driven tests usually involving the browser

INTRODUCTION

Automated End-to-End Testing of Web Applications has been growing for the last 10+ years. As of 2019, Selenium WebDriver and its Wire Protocol have been established as the industry standard for performing browser-driven acceptance tests for web applications. Given the fact that the tool can be accessed with virtually any popular programming language, a question can be raised: Which of the available language bindings enables the fastest execution for a typical web application? The goal of this paper was to compare two of the leading technology stacks: Java with the Selenide framework, and Ruby with the Watir framework, both frameworks being "wrappers" for pure Selenium, using a real application and two proof-of-concept testing frameworks, constructed for the purpose of this study.

The conclusion was that for three of the most popular internet browsers – Google Chrome, Mozilla Firefox and Internet Explorer, the only one for which there was statistically significant difference in test execution time was Internet Explorer. However, due to the fact that Internet Explorer has become almost obsolete as of the time of writing this paper, the conclusion was that no significant difference in execution time can be observed between Java and Ruby and therefore there are more important factors that come into play when choosing the right technology for implementing automated end-to-end acceptance tests of web applications.

The paper is structured in the following way:

Chapter one provides a broad overview of the automated testing field, introduces the concept of white- and black-box testing and aims at familiarizing the reader with the current research on the topic of automated testing.

Chapter two delves deeper into the technologies used for creating automated acceptance tests using two proof-of-concept projects created for the purpose of this study as examples.

Chapter three focuses on the execution of tests by the two frameworks.

Chapter four elaborates on the findings of the study.

Chapter five provides the summary for the thesis.

1 CHAPTER 1: THE CONCEPT OF AUTOMATED SOFTWARE TESTING

1.1 Automated Testing – definitions

According to Shaukat et. al. "*testing is an ongoing activity that is related with any process to produce a quality or working product. According to IEEE, Testing is the process of evaluating a system of system component by manual or automated means to verify that it satisfies required requirements. So, it is used to check the status of the working product after and during the software build. Software testing is also used to detect and identify the defects that the software may have. It is one of the vital parts of software development life cycle (SDLC)*"[1].

In the paper written by Bertolino[2], his author estimates that software testing is a discipline which started in 1970s. However, it could be argued that the first test ever performed was unofficial and done by the developer way earlier. According to Hetzel the first testing conference took place in 1972[3].

"Test automation is considered crucial for delivering the quality levels expected by users, since it can save a lot of time in testing and it helps developers to release web applications with fewer defects"[4]

"It is widely believed that approximately half the budget spent on software projects is spent on software testing, and therefore, it is not surprising that perhaps a similar proportion of papers in the software engineering literature are concerned with software testing"[5]

---

[1] Shaukat, K., Shaukat, U., Feroz, F., Kayani, S., & Akbar, A. (2015). Taxonomy of automated software testing tools. International *Journal of Computer Science and Innovation*, 1, p.7.

[2] Bertolino, A. (2007, May). Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering (pp. 85-103). *IEEE Computer Society*.

[3] Gelperin, D., & Hetzel, B. (1988). The growth of software testing. Communications of the ACM, 31(6), 687-695.

[4] Mlynarski M., Engels G. (2012) Model-Based Testing: Achievements and Future Challenges, Abstract

[5] Harman, M., Jia, Y., & Zhang, Y. (2015, April). Achievements, open problems and challenges for search based software testing. In 2015 IEEE *8th International Conference on Software Testing, Verification and Validation (ICST*) (pp. 1-12). IEEE. https://ieeexplore.ieee.org/abstract/document/7102580, Rerived 0n 20 of June 2019.

According to Dustin et. al Automated testing refers to *"The management and performance of test activities, to include the development and execution of test scrips so as to verify test requirements, using an automated test tool.*[6]*"*

1.2 Automated Testing Implementation success/fail factors

"Software testing can be done either by using automated or manual testing. By testing software through automated means is the best way to test the software. This testing of software is useful when repeated test scripts are required or where the test scripts subroutine are generated. The one of the most important advantage of automation testing is its execution speed. On the other hand, manual testing requires testing manually which needs more time, more chances of error and is no more useful[7]*"*

Garousi and Mäntylä list 42 of factors which should be considered when deciding on implementing automated testing solutions in an IT project . Out of them three were:

1. *"Our test engineers have adequate skills for test automation."*
2. *"We can afford to train our test engineers for test automation."*
3. *"We have expertise in the test automation approach and tool we have chosen"*[8]

1.3 Black-box and white-box testing

Software testing is a resource-intensive endeavor, yet it is an indispensable part of software engineering efforts. It is during this stage of the software development cycle the general quality of software is assessed and product defects are discovered. Traditionally, two main approaches have been used to test computer programs. The first one is called white-box testing. White-box testing means testing while knowing the internal workings of the system under test.[9] The second technique used widely in software testing is black-box testing, which by analogy, means testing the system without knowing its internal structure or

---

[6] Dustin et.al (in) E., Rashka, J., & Paul, J. (1999*). Automated software testing: introduction, management, and performance.* Addison-Wesley Professional., p. 3

[7] 5.Shaukat, K., Shaukat, U., Feroz, F., Kayani, S., & Akbar, A. (2015). Op.cit… p.7.

[8] Garousi, V., & Mäntylä, M. V. (2016). A systematic literature review of literature reviews in software testing. *Information and Software Technology*, 80, 208.

[9] http://glossary.istqb.org/search/white-box, retrieved 20.06.2019

implementation details, based solely on the system's specification i.e. the claims on how the system is supposed to work.[10]

White-box testing techniques include: unit testing - executing parts of the code in isolation and then comparing the results with expected behaviors, and integration testing - executing whole modules to see how they interact with one another.

Executing the entire system to check its behavior altogether falls under black-box testing, since it is done based on the product's specification.

While white-box testing has historically been performed in an automated way, black-box testing efforts had to be carried out manually.

However, due to the ever-accelerating technological progress, it is now possible to perform black-box testing in an automated fashion, with very little human oversight. Automated black-box tests are especially useful in the case of checking against new bugs introduced in the maintenance phase of the software lifecycle. The so-called regression testing of large systems involves executing hundreds of test cases and can be a repetitive and time-consuming process, which in turn can lead to high cost and the risk of errors on the part of the person executing tests due to fatigue.

Automated regression testing can free up testers' time allowing them to focus on more intricate and specialized testing.

Automated testing of web applications can be performed using various testing frameworks, but it is Selenium that is the golden standard of automated GUI testing these days. Selenium is a library which communicates with the internet browser and controls it as if it were being controlled by a real user. It mimics the actions a real user would be able to perform, such as visiting URLs (e.g. "google.com"), clicking on DOM elements on the page (click "Log In") and dragging and dropping objects, as well as taking screenshots.[11]

Selenium is a tool which provides a minimalistic API to drive the browser, but by itself it would be insufficient to perform automated testing. In order to compare the data retrieved by

[10] http://glossary.istqb.org/search/black-box, retrieved 20.05.2019
[11] https://www.seleniumhq.org/, retrieved on January 13, 2018

Selenium with expected results and to enable detailed reporting of test runs, additional libraries are needed.

There are multiple out-of-the-box tools which facilitate the aforementioned capabilities. Commercial tools include Ranorex, HP UFT and TestComplete. The two leading non-commercial solutions are cypress.io and Katalon Studio. Many tools however, even the commercial ones, make use of the Selenium protocol to drive the browser nonetheless, so it is safe to say Selenium has established itself as the default library for Black-Box Automated Testing.

1.4 Selenium

A variety of tools exist for testing applications. Now testing through automation tool has become the necessity of today era. Because not only does it save time, but it also saves the manual effort required to test the software for errors. The selection of tools requires the requirement gathering and to categorize them in a well manner.

According to Shaukat et.al (2015) (p.10102), there are at least 32 best tools that are automated tools nowadays. These tools include Selenium, Ranorex, Soap UI, FitNessee, Appvance, Apache J Meter, HP QTP, Test Complete and Telerik Test Studio.

Selenium is probably the most popular open source test automation framework for web applications. Selenium consists of three tools Selenium IDE, Selenium Web Driver and Selenium Grid. Selenium is a tool that is easy to use and freely available. Selenium supports multiple browsers. All components of the project are released under open source license Apache 2.0. Jason Huggins developed this tool in 2004. Nowadays Selenium has made a huge improvement as a strong web based automated tool.

1.5 Test execution time

Long test execution can lead to frustration on part of developers who must wait after each build for the tests to complete to check if their build has broken the application.

According to certain studies (Rogers, 2004), executing the whole test suite in 2 minutes would be the goal, because the longer it takes for a build to complete, the less frequently applications

get updated[12]. However, in the case of E2E tests it might never be a feasible goal. In author's experience, executing the whole test suite of Selenium tests takes hours, and it order to speed the process up so much so that it would take minutes, vast resources would have to be employed (executing all tests in parallel on the same machine where the SUT is hosted to mitigate network latency effects).

Test execution does not require human labor, but after each test run the results must be analyzed by a human in case one or more tests failed, if they are to be useful to the team. It would follow therefore that in order to save money on test automation better reporting could be devised – to save the time needed for a person to access the severity of the fault, or quicker tests.

According to studies at Microsoft, it is hard to tell precisely how much money can be saved by speeding up test execution vs. how much money could be saved by reducing the time needed to analyze test execution results. One study indicated test execution time is the biggest factor here, but another showed it was the quality of reporting.

In another article written by Kuutila et al. authors estimated that the greatest savings would be achieved by reducing the number of false-positively failing tests (tests failing where there is no real fault), and not by reducing test execution time[13].

One way of reducing execution time would be to restructure the tests in such a way that all tests run on the same instance of the browser. Currently, each test scenario begins with opening a new browser instance and ends with closing it. Opening a new process for a browser is time-consuming and it would seem like leaving just one browser open for all tests would be a better option. However, having multiple browser instances, one per each test scenario, enables parallel execution of the test suite. Parallel execution makes for better scalability of the test suite, because if tests take too long, more machines could be used to execute the test suite, which would reduce the total test execution time.

---

[12] Rogers, R.O., 2004. Scaling Continuous Integration, in: Extreme Programming and Agile Processes in Software Engineering, Vol. 3092, *The Series Lecture Notes in Computer Science*. pp. 68–76. doi:10.1007/978-3-540-24853-8_8

[13] Kuutila, M., Mäntylä, M., & Raulamo-Jurvanen, P. (2016). Benchmarking Web-testing-Selenium versus Watir and the Choice of Programming Language and Browser. arXiv preprint arXiv:1611.00578.

1.6 The aim of the project: An automated testing solution for Redmine

The aim of this project is to compare the performance of the Selenium webdriver library for two object-oriented programming languages: Ruby and Java. Java has been the leading technology for creating automated tests in the browser, but there are proponents of other competing technologies and Ruby, being a well-established object-oriented language, is one of them.

The aim of this project is to establish which technology stack – Java or Ruby – is better suited for creating Black-Box Automated Testing frameworks using Selenium. The main criterion used to compare the two will be the execution time for three most popular browsers on Windows – Google Chrome and Mozilla Firefox and Internet Explorer.

As of the time of writing this thesis, five of the most popular programming languages have been supported by the Selenium project. They are: Java, C#, Ruby, Python and Javascript. The most popular language binding for Selenium is Java, but judging by job listings and internet discussions, Python, Ruby and Javascript have their fair share of popularity among Test Engineers worldwide, too.

It is not clear, however, which of the five officially supported language bindings for Selenium could be considered "the best" for test automation. Java has the biggest community of users among Test Engineers, but other languages have their advantages too.

Ruby is considered the slower language compared to Java, but it is not obvious how much slower it is and if it is slower in the narrow context of running a suite of Selenium tests. Knowing which Selenium binding works faster comes with real-life monetary implications. The main advantage Ruby has over Java is that its syntax is terser (in general the same snippet of code written in Java requires more characters than expressed in Ruby).

In order to evaluate the performance of the Ruby and Java-based projects mean execution time from the very beginning to the end of a run will be measured using the Cucumber test runner in conjunction with Allure Reports, a graphical reporting tool. Both tools are open-source and there are implementations for Java and Ruby available.

A secondary benchmark used to compare the two technologies will the number of lines of code for each project as well as the total number of files in each project.

The Application under Test used for this comparison will the Redmine Project Management application.

The test suites for both languages will cover the same basic functionalities of Redmine. I started off by creating a test suite in Java and then I tried to replicate the tests as closely as possible in Ruby.

Both Ruby and Java projects use Cucumber which is a test runner utilizing the Behavior Driven Development methodology. BDD is based around the separation of test specification (.features text files) and actual code to execute assertions and arrangements (written in Java or Ruby).

Developing the Ruby-based framework proved to be more difficult since there were fewer working examples of successful attempts at combining all the necessary components for the framework. Java comes with a better dependency management environment – Maven – and the examples found on the web worked more consistently.

1.7 Conclusions from literature overview and future trends in research

Google Scholar lists 16400 scientific papers dealing with Automated Web Testing with Selenium have been published since the tool's inception in 2004[14].

However, according to Vahid Garousi and Mika Mäntylä, there is very little written about when and what to automate in testing. Most of the publications are so called grey literature and are not backed by research[15]. Most of publications on automated testing are materials and research produced by organizations outside of the traditional commercial or academic publishing. "*The majority of prior works are opinion or experience papers that provide heuristics or guidelines while only a small share of papers present more rigorous work of evaluation or validation research presenting processes or models*"[16]. Garousi and Mantayla underline that it as a pity that practitioners do not want to publish and share their experience with scientists. From their synthesis of literature it can be concluded that only 14% of studied

[14] , https://scholar.google/ retrieved on 24. 06.2019
[15] Garousi, V., & Mäntylä, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, 76, 92-117.
[16] Garrousi et. al., op.cit.

literature provides evidence on the topic. Which means that "a lack of high-quality real-world studies of this topic exists"[17]. This means that there is a need for intensification of scientific work and scientific studies especially directed on modern development concepts such Agile software development[18], DevOps[19] and Continuous Deployment [20]. Similarly, Arcuri (2018) states that there are gaps in research practices in software engineering and particularly in software testing,[21].

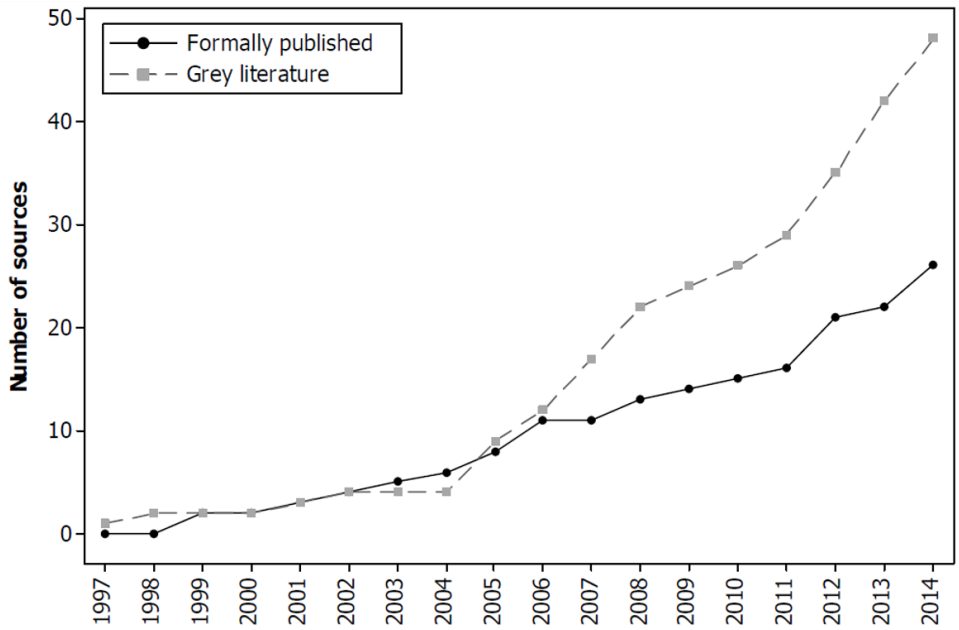

Figure 1: Cumulative number of sources per year, "When and what to automate in software testing? A multi-vocal literature" 2016 by V. Garousi, M. Mäntylä

Source: Garousi, V., & Mäntylä, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology, 76, p. 11.

[17] Ibidem p. 28.
[18] Hoda, R., Salleh, N., & Grundy, J. (2018). The rise and evolution of agile software development. IEEE Software, 35(5), 58-63.
[19] Laukkarinen, T., Kuusinen, K., & Mikkonen, T. (2018). Regulated software meets DevOps. *Information and Software Technology*, 97, 176-178.
[20] Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, J. (2018). Requirements engineering and continuous deployment. *IEEE software*, 35(2), 86-90.
[21] Arcuri, A. (2018). An experience report on applying software testing academic results in industry: We need usable automated test generation. *Empirical Software Engineering*, 23(4), 1959-1981.

2 CHAPTER 2: TECHNOLOGIES USED IN THE PROJECT

2.1 Two frameworks

Two complete testing frameworks have been created for the purpose of running Selenium tests. One of them has been written in Java, the other one in Ruby. They share the same Cucumber .feature files, which define test steps in terms of business cases, but each of them implements the test steps defined in those files in the way specific to the language they were written in. Because of the way Selenium has been architectured, both frameworks end up sending the same API calls to one of three WebDriver HTTP servers (one for Chrome, one for Firefox, and one for Internet Explorer). Those calls are in turn translated into clicks and other actions performed by the browser and then feedback information is sent back to the WebDriver server and back to the testing framework where checks are performed to compare the data expected by the test with the data received from the browser. The outcome of those checks is saved by the frameworks after each test and after all tests are done, a report is generated to present test results in a human-readable format.

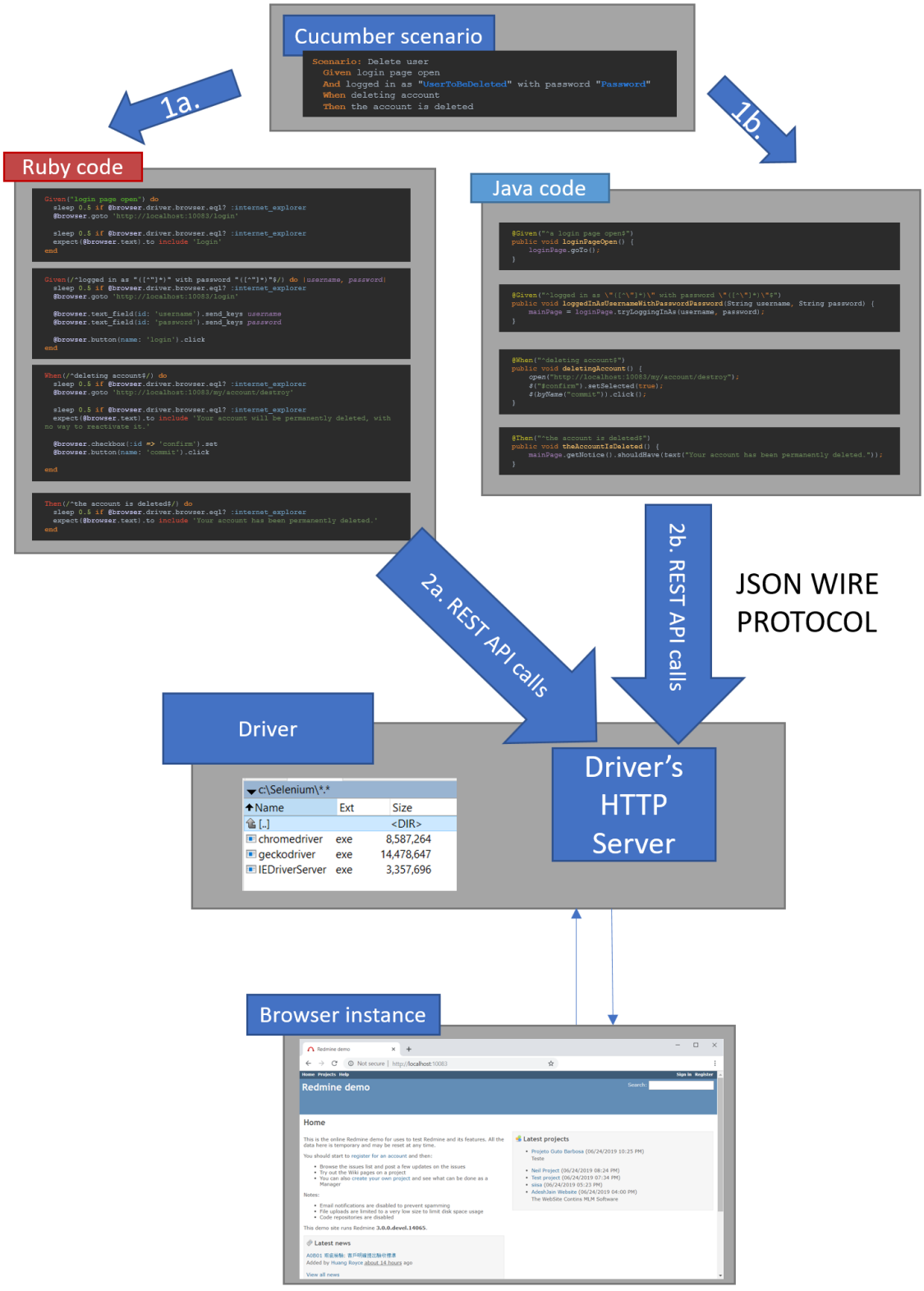

Figure 2: Proof-of-concept test framework's information flow

Source: Own elaboration

1a. Cucumber for Java looks for step definitions in the Java projects and executes them in the order specified in the .feature files.

1b. Cucumber for Ruby looks for step definitions in the Ruby projects and executes them in the order specified in the .feature files.

2a. Selenide (Java) method calls are translated into Selenium method calls. Selenium method calls result in HTTP requests sent to an instance of a webdriver server using the Selenium Wire Protocol

2b. Watir (Ruby) method calls are translated into Selenium method calls. Selenium method calls result in HTTP requests sent to an instance of a webdriver server using the Selenium Wire Protocol

3. Each driver communicates with an instance of the browser they drive in a way specific to it

Table 1: Ruby and Java projects comparison

	Java-based project	Ruby-based project
Programming language	Java is a general-purpose programming language, widely used in the corporate world. It is the most popular language used for developing Black-Box Automated Testing frameworks (BATs) which use the Selenium webdriver.[22]	Ruby is an object-oriented modern programming language, but unlike Java it has mostly been used for developing web applications leveraging the Ruby on Rails development framework. There are less downloads of the Selenium binding for Ruby than for Java and quick google search reveals it is less popular for creating BATs than Java.
Testing framework	Selenide Selenide is a wrapper framework for Selenium. It allows the user to use a terser syntax to communicate with the browser. Terser that it would be if the code were written using the Selenium API. It can also download the drivers for Chrome, Firefox and Internet Explorer automatically.	Waitr Watir is a wrapper framework for Selenium, much like Selenide is for Java. It provides one more abstraction layer for the programmer to make things easier when using Selenium. Before the release of Watir 6.0, Watir used its own driver implementations, but since 2016 Watir takes advantage of Selenium drivers.[23]
Test runner	Cucumber Cucumber is test runner used for BDD-style tests. Behavior-Driven Development is a methodology for organizing the development and testing in IT projects. In BDD, acceptance criteria for features are written	

[22] Test Talks Podcast with Dave Haefffner – Episode 85
https://www.joecolantonio.com/testtalks/85-dave-haeffner-the-java-selenium-guidebook/,
03:00, "This year [2016] Java owns 65% of downloads for Selenium."
[23] http://watir.com/history/, retrieved 20.05.2019.

	in a predefined way using the Gherkin notation. Keywords such as Given, When, and Then are used to specify test steps and expected results in a way that is easy to understand by non-technical counterparties in the project.
Reporting tool	Allure Reports Allure Reports is a reporting tool developed by the Russian internet company Yandex. It is virtually language-agnostic and helps visualize test results in an aesthetically pleasing way. There are implementations for Ruby and Java, so the end report will look very similar for Java and Ruby, which will make comparing the results for both languages easier.

Source: own elaboration

2.2 Other technologies used in research

Docker. Docker is a platform which lets its users replace tedious installation steps with a few commands in CLI by leveraging OS-virtualization and containers. It has been used to spin up Redmine on the same computer as the test frameworks.[24][25]

[24] https://pdfs.semanticscholar.org/fd3d/f1574e56997e5064e749a7111451f6ae0cab.pdfe, retrieved 20.05.2019
[25] https://www.docker.com/resources/what-container, retrieved 20.05.2019

3 CHAPTER 3: METHODOLOGY AND CONFIGURATION

3.1 The technical specifications of the computer the tests were executed on and software versioning

Table 2: The technical specifications of the computer the tests were executed on and software versioning

Computer	
Windows 10	
CPU Intel Core i7 i7-8550U 1.80GHz	
16 GB RAM	
Internet browsers	WebDrivers
Chrome version 74.0.3729.169 (64-bit)	chromedriver.exe version 74.0.3.3729.6
Firefox version 67.0.2 (64-bit)	geckodriver.exe version 0.24.0
Internet Explorer version 11.829.17134.0	IEDriverServer.exe version 3.141
Ruby project:	Java project:
Watir version 6.16.5	Selenide version 5.1.0
Cucumber for Ruby version 3.1.2	cucumber-java 3.0.2
allure-cucumber 0.6.1	allure-maven 2.9
allure-ruby-adaptor-api 0.7.2	Allure-cucumber3-jvm 2.7.0

Source: own elaboration

3.2 Browsers

According to GlobalStats, the most popular desktop browsers used worldwide in 2019 were: Chrome (69.09% market share), Firefox (10.01% market share), Safari (7.25%) and Internet Explorer (5.14%). Since Safari runs only on Apple computers, it has been decided that only Chrome, Firefox and IE would be used in the project.

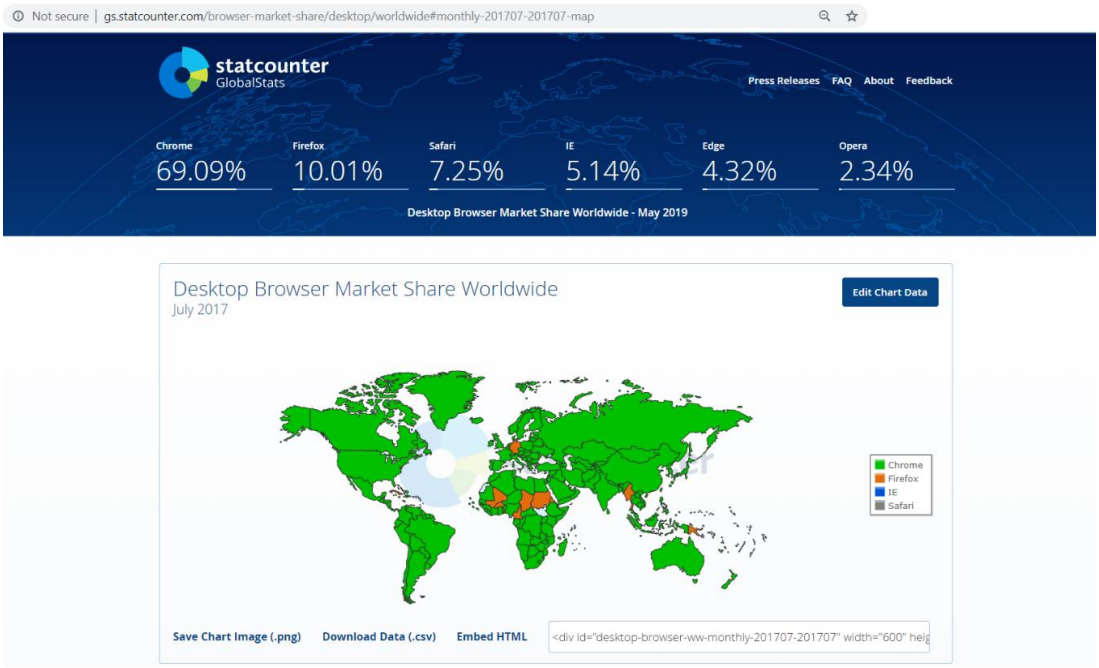

Figure 3: Internet Browser market share in 2019

Source: http://gs.statcounter.com/browser-market-share/desktop/worldwide#monthly-201707-201707-map, retrieved on June 10, 2019

3.3 Test cases

The test suit created for this experiment consists of 12 test cases. They test Redmine's most rudimentary capabilities. The test cases are:

1. Create an issue and log time: Creating and issue and logging time
2. Create and deleting private projects: Creating a new project, setting it to private, trying to access project without permission
3. Create and deleting private projects: Deleting a project as admin
4. Delete user: Delete user
5. Login: Logging in as a user which exists
6. Login: Logging in with incorrect credentials
7. Projects: Closing a project (making it read-only)
8. Projects: Creating a new project
9. Projects: Reopening a project
10. Registration: Registering a new user

11. Registration: Trying to register a user which already exists
12. Update project wiki (file upload): Upload file

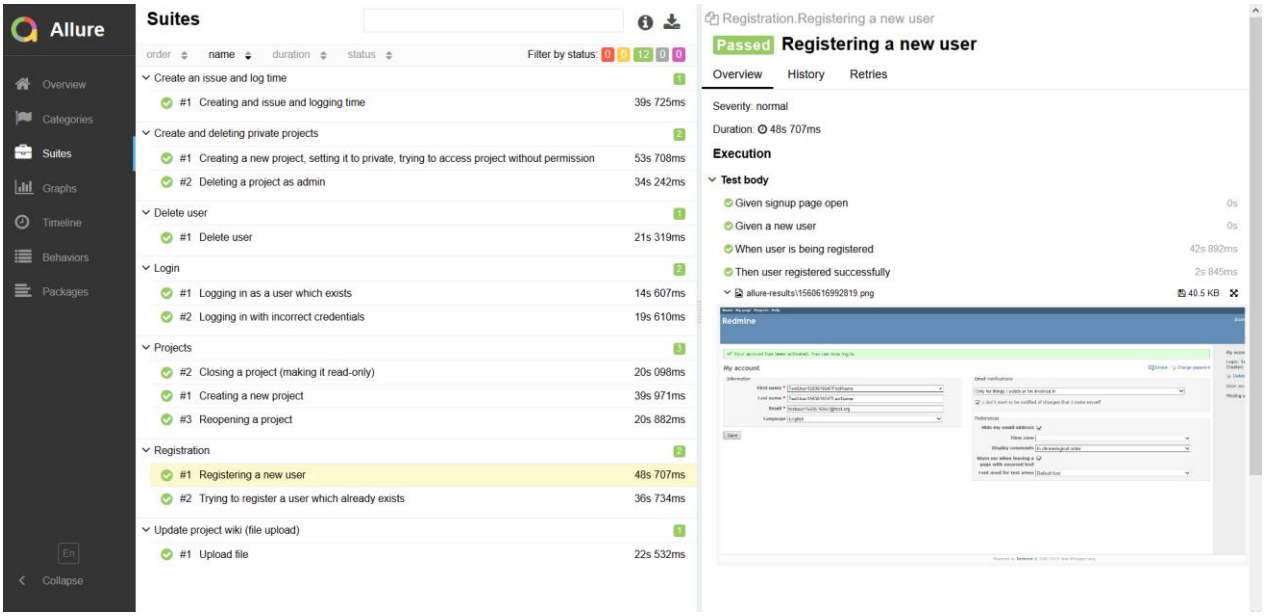

Figure 4:Allure Report - all test cases

Source: own elaboration

3.4 Test setup

Before each test suite run, a test setup script was executed. In order to put Redmine in the state necessary for tests, the Selenium driver has been leveraged to set up projects and user accounts needed for each test run.

```
#!/usr/bin/env bash

export BROWSER_TYPE=chrome;
rm allure-results allure-report -rf;
cucumber  --tags "@test_setup" --format AllureCucumber::Formatter --out allure-results;
```

Figure 5: Test setup bash script
source: own code

The run_java_generate.sh script would first run the test_setup.sh and then execute a Maven command to run the JavaRedmineAutomation project. The last step would be to copy the execution report of the run to a designated folder and rename it using the convention “language_browser_MonDD_HH-MM” eg. “Java_chrome_Jun15_08-17PM”.

In case of run_ruby_generate.sh script the only difference would be running the project via the cucumber command instead of Maven, since Maven is only used for Java.

In order to measure the execution time of automated tests, a System Under Test was needed.

The initial approach for measuring execution times was to use an external application, hosted on the Internet.[26] However, it turned out that during certain times of the day, the execution took longer than at other times, even for the same browser and language binding. In author's view the explanation was that Redmine Demo was accessed by users around the world at different times and this varying load impacted the application's performance.

After the initial fiasco, a different approach was chosen. In order to isolate test results from external issues such as internet connectivity and SUT's response times which vary due to unpredictable load, Redmine has been installed on the same machine as the test executors.

All tests from now on would be executed against Redmine 4.0.3-1 spun locally using a docker image [27] application, which ran on localhost, port 10083.

[26] http://demo.redmine.org, retrieved on January 10, 2019.
[27] https://github.com/sameersbn/docker-redmine , retrieved 20.05.2019.

4 CHAPTER 4: FINDINGS

4.1 Execution times

Tests results have been analyzed using the two-way ANOVA[28] method and no significant difference in execution times has been found between Ruby and Java for Automated GUI Tests in Selenium for Chrome and Firefox. However, there was a noticeable execution time disparity between the two programming languages with regards to tests run in Internet Explorer 11.

Figure 6: Test Execution Time for Java and Ruby

Source: own research

[28] Two-way ANOVA models with unbalanced data, Yasunori Fujikoshi, 1993

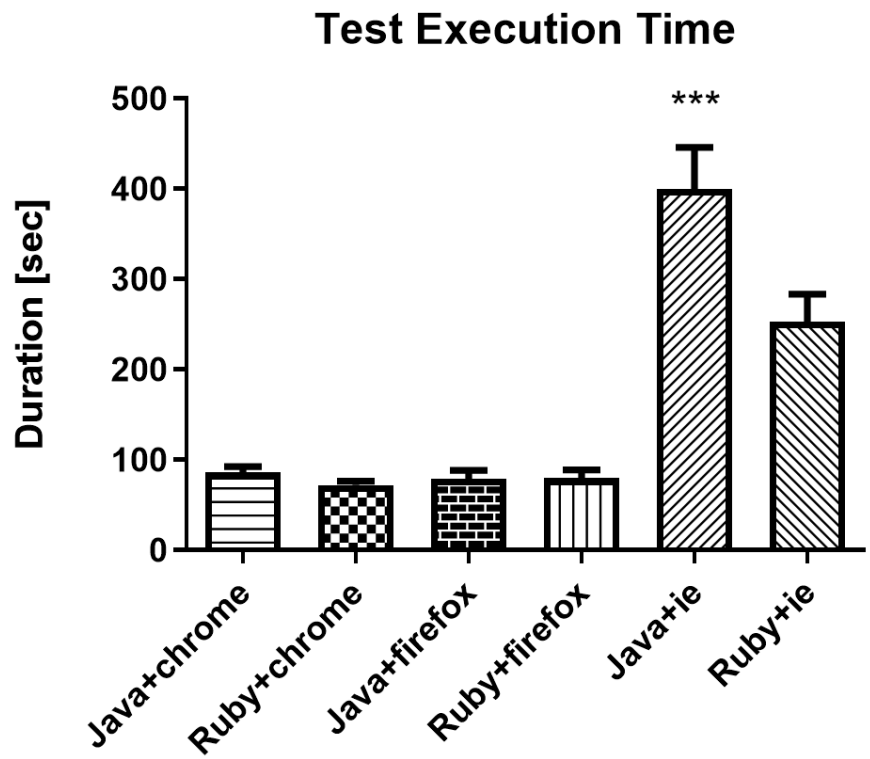

Figure 7: Test execution time for all browser-language combinations

Source: own research

Table 3: Test execution time measured for all browser-language combinations

	Java			Ruby		
	Mean	SD	N	Mean	SD	N
Chrome	81.52533	19.34137	12	72.20808	15.06056	12
Firefox	79.57136	30.25216	11	79.84792	31.98256	12
IE11	400.229	152.0117	11	253.1561	105.4749	12

Source: own research

Table 4: All test runs

#	Test run	language	browser	duration [sec]
1	Java_chrome_Jun15_05-44PM	Java	chrome	124.793
2	Java_chrome_Jun15_08-17PM	Java	chrome	109.338
3	Java_chrome_Jun15_09-12PM	Java	chrome	70.504
4	Java_chrome_Jun16_03-19PM	Java	chrome	71.791

5	Java_chrome_Jun16_03-42PM	Java	chrome	71.511
6	Java_chrome_Jun16_04-01PM	Java	chrome	71.368
7	Java_chrome_Jun16_04-18PM	Java	chrome	71.001
8	Java_chrome_Jun16_04-36PM	Java	chrome	71.094
9	Java_chrome_Jun16_04-54PM	Java	chrome	70.855
10	Java_chrome_Jun16_05-12PM	Java	chrome	71.243
11	Java_chrome_Jun16_05-30PM	Java	chrome	70.95
12	Java_chrome_Jun16_08-23PM	Java	chrome	103.856
13	Java_firefox_Jun15_05-36PM	Java	firefox	138.738
14	Java_firefox_Jun15_09-08PM	Java	firefox	75.232
15	Java_firefox_Jun16_03-15PM	Java	firefox	96.528
16	Java_firefox_Jun16_03-38PM	Java	firefox	62.821
17	Java_firefox_Jun16_03-57PM	Java	firefox	64.273
18	Java_firefox_Jun16_04-14PM	Java	firefox	61.112
19	Java_firefox_Jun16_04-32PM	Java	firefox	60.053
20	Java_firefox_Jun16_04-50PM	Java	firefox	60.443
21	Java_firefox_Jun16_05-08PM	Java	firefox	59.582
22	Java_firefox_Jun16_05-26PM	Java	firefox	61.884
23	Java_firefox_Jun16_08-16PM	Java	firefox	134.619
24	Java_ie_Jun15_05-56PM	Java	ie	463.028
25	Java_ie_Jun16_02-42PM	Java	ie	629.495
26	Java_ie_Jun16_03-01PM	Java	ie	269.962
27	Java_ie_Jun16_04-08PM	Java	ie	286.206
28	Java_ie_Jun16_07-28PM	Java	ie	295.588
29	Java_ie_Jun16_07-35PM	Java	ie	297.219
30	Java_ie_Jun16_07-45PM	Java	ie	307.049
31	Java_ie_Jun16_07-53PM	Java	ie	307.168
32	Java_ie_Jun16_07-59PM	Java	ie	306.988
33	Java_ie_Jun16_08-11PM	Java	ie	635.447
34	Java_ie_Jun16_08-38PM	Java	ie	604.369
35	Ruby_chrome_Jun15_05-47PM	Ruby	chrome	86.684
36	Ruby_chrome_Jun15_07-12PM	Ruby	chrome	89.957
37	Ruby_chrome_Jun15_07-34PM	Ruby	chrome	95.673
38	Ruby_chrome_Jun15_09-14PM	Ruby	chrome	61.787
39	Ruby_chrome_Jun16_03-21PM	Ruby	chrome	61.824
40	Ruby_chrome_Jun16_03-44PM	Ruby	chrome	62.335
41	Ruby_chrome_Jun16_04-03PM	Ruby	chrome	61.863
42	Ruby_chrome_Jun16_04-20PM	Ruby	chrome	62.202
43	Ruby_chrome_Jun16_04-38PM	Ruby	chrome	62.726
44	Ruby_chrome_Jun16_04-56PM	Ruby	chrome	62.571
45	Ruby_chrome_Jun16_05-14PM	Ruby	chrome	61.952
46	Ruby_chrome_Jun16_08-26PM	Ruby	chrome	96.923
47	Ruby_firefox_Jun15_05-40PM	Ruby	firefox	135.787
48	Ruby_firefox_Jun15_07-08PM	Ruby	firefox	133.348
49	Ruby_firefox_Jun15_07-29PM	Ruby	firefox	124.943

50	Ruby_firefox_Jun15_09-10PM	Ruby	firefox	76.527
51	Ruby_firefox_Jun16_03-17PM	Ruby	firefox	78.539
52	Ruby_firefox_Jun16_03-40PM	Ruby	firefox	59.89
53	Ruby_firefox_Jun16_03-58PM	Ruby	firefox	58.927
54	Ruby_firefox_Jun16_04-16PM	Ruby	firefox	57.771
55	Ruby_firefox_Jun16_04-34PM	Ruby	firefox	57.903
56	Ruby_firefox_Jun16_04-52PM	Ruby	firefox	57.603
57	Ruby_firefox_Jun16_05-10PM	Ruby	firefox	58.533
58	Ruby_firefox_Jun16_05-28PM	Ruby	firefox	58.404
59	Ruby_ie_Jun15_06-04PM	Ruby	ie	382.748
60	Ruby_ie_Jun15_07-22PM	Ruby	ie	396.312
61	Ruby_ie_Jun15_07-44PM	Ruby	ie	406.756
62	Ruby_ie_Jun15_09-22PM	Ruby	ie	177.623
63	Ruby_ie_Jun16_03-31PM	Ruby	ie	183.763
64	Ruby_ie_Jun16_03-54PM	Ruby	ie	183.387
65	Ruby_ie_Jun16_04-12PM	Ruby	ie	181.818
66	Ruby_ie_Jun16_04-30PM	Ruby	ie	182.272
67	Ruby_ie_Jun16_04-48PM	Ruby	ie	182.204
68	Ruby_ie_Jun16_05-06PM	Ruby	ie	182.062
69	Ruby_ie_Jun16_05-24PM	Ruby	ie	181.606
70	Ruby_ie_Jun16_08-46PM	Ruby	ie	397.322

Source: own research

4.2 Project size and test design

In addition to execution times, project sizes have been measured using cloc[29], an open-source tool. The project created in Java contains almost three times as many lines of code as the one in Ruby. Java is known for its verbosity, but another reason behind this state of things might be a decision behind the design of each project. Tests written in Java employ the so-called Page Object Pattern encouraged by the authors of Selenide. The pattern involves creating separate objects representing pages of the application under test, and then using the references to those pages in tests, without referring to elements on the screen directly, by locators. In the case of Ruby and Watir, the pattern was not utilized, which may have influenced the total count of lines of code in each project.

```
package com.practice.redmine.automation.steps;

import com.practice.redmine.automation.pages.LoginPage;
import com.practice.redmine.automation.pages.MainPage;
import com.practice.redmine.automation.utils.UserDataGenerator;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
```

[29] https://github.com/AlDanial/cloc, retrieved 20.05.2019.

```java
import java.io.IOException;

import static com.codeborne.selenide.Condition.text;
import static com.codeborne.selenide.Selectors.byName;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.open;
import static org.junit.Assert.assertEquals;
import static org.junit.Assume.assumeFalse;

public class DeleteTestUsersSteps {

    LoginPage loginPage = new LoginPage();
    MainPage mainPage;

    @Given("^a login page open$")
    public void loginPageOpen() {
        loginPage.goTo();
    }

    @Given("^logged in as \"([^\"]*)\" with password \"([^\"]*)\"$")
    public void loggedInAsUsernameWithPasswordPassword(String username, String password) {
        mainPage = loginPage.tryLoggingInAs(username, password);
    }

    @When("^deleting account$")
    public void deletingAccount() {
        open("http://localhost:10083/my/account/destroy");
        $("#confirm").setSelected(true);
        $(byName("commit")).click();
    }

    @Then("^the account is deleted$")
    public void theAccountIsDeleted() {
        mainPage.getNotice().shouldHave(text("Your account has been permanently deleted."));
    }
}
```

Figure 8: An example of test steps implemented in Java

source: own code

```ruby
include AllureCucumber::DSL

When(/^deleting account$/) do
  sleep 0.5 if @browser.driver.browser.eql? :internet_explorer
  @browser.goto 'http://localhost:10083/my/account/destroy'

  sleep 0.5 if @browser.driver.browser.eql? :internet_explorer
  expect(@browser.text).to include 'Your account will be permanently deleted, with no way to reactivate
it.'

  @browser.checkbox(:id => 'confirm').set
  @browser.button(name: 'commit').click

end

Then(/^the account is deleted$/) do
  sleep 0.5 if @browser.driver.browser.eql? :internet_explorer
  expect(@browser.text).to include 'Your account has been permanently deleted.'
end
```

Figure 9:An example of test steps implemented in Ruby

source: own code

Table 5: Number of test lines for Ruby

```
$ cloc ruby-redmine-automated-tests/
      29 text files.
      29 unique files.
      11 files ignored.

github.com/AlDanial/cloc v 1.82  T=0.50 s (46.0 files/s, 1398.0 lines/s)
```

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Ruby                            10            162              6            348
Cucumber                         8             25              0            114
Bourne Shell                     4             14              0             29
YAML                             1              0              0              1
-------------------------------------------------------------------------------
SUM:                            23            201              6            492
-------------------------------------------------------------------------------
```

Source: own research

Table 6: Number of test lines for Java

```
$ cloc JavaRedmineAutomation
      51 text files.
      51 unique files.
      28 files ignored.

github.com/AlDanial/cloc v 1.82  T=1.00 s (47.0 files/s, 1836.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                            35            353            201           1002
Cucumber                         8             25              0            113
Maven                            1             12              4             90
Markdown                         1              3              0             15
Bourne Shell                     2              7              0             11
-------------------------------------------------------------------------------
SUM:                            47            400            205           1231
-------------------------------------------------------------------------------
```

Source: own research

SUMMARY

Interpretation of results

The differences in execution times for Chrome and Firefox between Java and Ruby are negligible. In the case of Internet Explorer 11, Ruby was 1.6 faster on average than Java, but version 11 was the last version and IE is no longer maintained by Microsoft. In author's view the results hint at the conclusion that there more important factors for deciding on the technology stack used for creating Automated E2E Tests for web applications than the sheer time of execution.

The results echo the opinion, voiced by various Test Automation experts, that the choice for the language of the test automation framework should be based on a variety of factors. In the view of those experts, neither Selenium binding could be considered "the best" and the decision should consider project practicalities e.g. what language is used for the System Under Test.[30]

The results revealed that there was no significance difference test execution time for Ruby and Java for Google Chrome and Mozilla Firefox browsers. These two are the most popular as of the time of writing this paper. However, for Internet Explorer 11 Ruby vastly outpaced Java, which might inform the choice of the programming language for a GUI-driven automated testing framework, if the System Under Test is mostly used by Internet Explorer 11 users. However, it is worth noting that Internet Explorer 11 is the last version of IE, therefore as years go by less and less people will be using it.

Future developments

However, even though the results showed no significant difference between the browsers and languages in terms of test execution speed for Chrome and Firefox, the tests could yield different results for other applications under tests for these two browsers. Redmine is not a Javascript-heavy application, it lacks animation and there aren't many (if any) asynchronous AJAX calls employed. It has been my experience testing other applications on the Internet,

[30] https://www.cigniti.com/blog/programing-language-to-build-selenium-based-test-automation-suite/ https://applitools.com/blog/language-software-test-automation\ https://www.joecolantonio.com/selenium-what-programming-language-you-should-learn-to-get-into-test-automation/

that Firefox tends to be slower than Chrome, however, in this particular case the execution times were comparable. Internet Explorer behaves differently when used with I think further investigation on a broader number of web application might reveal potentially valuable differences between the languages too.

Problems encountered while implementing automated tests in Java and Ruby

Working with Ruby proved to be more difficult than working with Java. It took many tries to finally integrate all project dependencies ("gems") and get the project up and running.

All tests except for "Update project wiki (file upload): Upload file" were developed for Demo Redmine[31]. After switching over to Redmine on localhost, two tests started failing. The so called "false positives" were present in those test cases which dealt with creating and disabling projects. After some fine-tuning it turned out the problem with "flaky" tests (passing and failing unreliably) was caused by insufficient wait time in test assertions.

It also took many tries to enable successful test execution on Internet Explorer, both for Ruby and for Java.

[31] http://demo.redmine.org., retrieved 12.05.2019

REFERENCES

Arcuri, A. (2018). An experience report on applying software testing academic results in industry: We need usable automated test generation. Empirical Software Engineering, 23(4), 1959-1981.

Bertolino, A. (2007, May). Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering (pp. 85-103). IEEE Computer Society.

Dustin, E., Rashka, J., & Paul, J. (1999). Automated software testing: introduction, management, and performance. Addison-Wesley Professional, USA.

Fujikoshi, Y. (1993). Two-way ANOVA models with unbalanced data. Discrete Mathematics, 116(1-3), 315-334.

Garousi, V., & Mäntylä, M. V. (2016). A systematic literature review of literature reviews in software testing. Information and Software Technology, 80, 195-216.

Gelperin, D., & Hetzel, B. (1988). The growth of software testing. Communications of the ACM, 31(6), 687-695.

Gojare, S., Joshi, R., & Gaigaware, D. (2015). Analysis and design of selenium webdriver automation testing framework. Procedia Computer Science, 50, 341-346.

Kuutila, M., Mäntylä, M., & Raulamo-Jurvanen, P. (2016). Benchmarking Web-testing-Selenium versus Watir and the Choice of Programming Language and Browser. arXiv preprint arXiv:1611.00578.

Laukkarinen, T., Kuusinen, K., & Mikkonen, T. (2018). Regulated software meets DevOps. Information and Software Technology, 97, 176-178.

Mlynarski, M., Güldali, B., Weißleder, S., & Engels, G. (2012). Model-based testing: achievements and future challenges. In Advances in Computers (Vol. 86, pp. 1-39). Elsevier.

Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, J. (2018). Requirements engineering and continuous deployment. IEEE software, 35(2), 86-90.

Rogers, R.O., 2004. Scaling Continuous Integration, in: Extreme Programming and Agile Processes in Software Engineering, Vol 3092 of the Series Lecture Notes in Computer Science. pp. 68–76. doi:10.1007/978-3-540-24853-8_8

Shaukat, K., Shaukat, U., Feroz, F., Kayani, S., & Akbar, A. (2015). Taxonomy of automated software testing tools. International Journal of Computer Science and Innovation, 1, 7-18.

Smart J.F. (2014) Java. Praktyczne narzędzia (ebook), Wydawnictwo: Helion

Yehezkel, S., 2016. Test Automation Survey 2016 [WWW Document]

http://allure.qatools.ru/

http://glossary.istqb.org/search/white-box, retrieved 20.06.2019

http://watir.com/history

https://docs.oracle.com/javase/8/docs/

https://www.seleniumhq.org/, as of January 13, 2018.Test Talks Podcast with Dave Haeffner – Episode 85 https://www.joecolantonio.com/testtalks/85-dave-haeffner-the-java-selenium-guidebook/ ,

https://www.ruby-lang.org/en/documentation

https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

LIST OF FIGURES

LIST OF TABLES

STRESZCZENIE PRACY INŻYNIERSKIEJ W JĘZYKU POLSKIM

Automatyczne testowanie aplikacji internetowych rozwija się od ponad 10 lat. Selenium WebDriver oraz Wire Protocol, którego Selenium używa, zostały ustanowione jako de-facto standard branżowy do przeprowadzania testów akceptacyjnych w przeglądarce. Biorąc pod uwagę fakt, że Selenium można obsługiwać z praktycznie dowolnego popularnego języka programowania, nasuwa się pytanie: który z dostępnych języków umożliwia najszybsze wykonanie testów typowej aplikacji internetowej? Celem tej pracy było porównanie dwóch wiodących stosów technologicznych do testów automatycznych aplikacji webowych: Java'y z frameworkiem Selenide oraz Ruby'iego z frameworkiem Watir.

Rozdział pierwszy zawiera przegląd literatury dotyczącej automatycznego testowania, wprowadza koncepcję testowania biało- oraz czarnoskrzynkowego oraz ma na celu zapoznanie czytelnika z aktualnymi badaniami na temat testów automatycznych.

Rozdział drugi zagłębia się w technologie wykorzystywane do tworzenia automatycznych testów akceptacyjnych na przykładzie dwóch projektów stworzonych na potrzeby tego badania.

Rozdział trzeci skupia się na wykonywaniu testów przez dwa projekty.

Rozdział czwarty omawia wyniki badania.

Rozdział piąty zawiera podsumowanie tezy.

Różnica w czasie wykonania dla przeglądarek Firefox oraz Chrome była statystycznie pomijalna. Dla przeglądarki Internet Explorer analiza two-way ANOVA wykazała znaczną różnicę dla obu języków. Testy w Rubym były średnio około 1,6 razy szybsze niż te same testy napisane w Javie. Nie mniej jednak, Internet Explorer nie jest już utrzymywany przez Microsoft, dlatego liczba użytkowników tej przeglądarki będzie zanikać. Mimo że wyniki nie wykazały znaczącej różnicy między przeglądarkami i językami pod względem szybkości wykonywania testów dla Chrome i Firefox, testy mogą dać różne wyniki dla innych aplikacji testowanych dla tych dwóch przeglądarek. Redmine nie jest aplikacją, która wykorzystuje intensywnie Javascript, zapytania asynchroniczne oraz nie posiada animacji. Zdaniem autora, badania szerszej liczby aplikacji internetowych mogą ujawnić potencjalnie różnice w czasach

wykonania testów między językami. Nie mniej jednak, bazując na wynikach badań przeprowadzonych dla celów tej pracy, w opinii autora wyniki wskazują na wniosek, że czasy wykonania dla Java i Rubiego są bardzo porównywalne i dlatego czas wykonania nie powinien być decydującym czynnikiem przy wyborze technologii do tworzenia automatycznych testów aplikacji webowych. Potrzeba większej ilości badań dotyczących testów automatycznych przy wykorzystaniu Selenium oraz zgłębienie czynników istotnych przy wyborze technologii do testowania automatycznego aplikacji.

SPIS TREŚCI W JĘZYKU POLSKIM

SPIS RYSUNKÓW

SPIS TABEL