

**Mendelova univerzita v Brně
Provozně ekonomická fakulta**

Automation of regression testing of web application

Bachelor thesis

Supervisor:

Ing. Pavel Haluza, Ph.D.

Bernard Badó

Brno 2016

I would like to thank my supervisor Ing. Pavel Haluza, Ph.D. for helpfulness, advices, suggestions and consultations. My thanks also belong to quality assurance team within the Engage department.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Automation of regression testing of web application**

vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací*.

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 17. května 2016

Abstrakt

Badó, B. Automatizácia regresného testovania webovej aplikácie. Bakalárska práca. Brno: Mendelova univerzita v Brně, 2016.

Cieľom tejto práce je naštudovať problematiku automatizácie regresného testovania a prostriedkov, ktoré sú k automatizácií využívané. Práca obsahuje popis webovej aplikácie, na ktorej bude automatizované testovanie demonštrované. V práci je ďalej predstavená sada automatizovaných testov, vyvíjaná spoločnosťou Blackboard Czech s.r.o v rámci projektu Engage QA automation, ktorá overuje funkčnosť spomínanej webovej aplikácie.

Kľúčové slová

Automatizované testovanie, regresné testovanie, webová aplikácia, selenium

Abstract

Badó, B. Automation of regression testing of web application. Bachelor thesis. Brno: Mendel University, 2016.

The aim of this thesis is to study automation of regression testing and its technologies, which are used for automation. It contains description of the web application, which is used to demonstrate automated tests. The work also contains examples of test suite developed by Blackboard Czech s.r.o within the Engage QA automation project, which validate functionality of the mentioned web application.

Keywords

Test automation, regression testing, web application, selenium

Table of Contents

1	Introduction and aim of this thesis	11
1.1	Introduction	11
1.2	Aim of this thesis	11
2	Literature search	12
3	Software testing	13
3.1	Testing methods	13
3.2	Testing levels	14
3.3	Types of testing	15
3.4	Testing process	16
3.4.1	Waterfall development model.....	17
3.4.2	Agile development model	18
4	Test automation tools	20
4.1	Advantages of test automation	20
4.2	Unit testing frameworks.....	21
4.3	Automated web testing tools.....	26
4.4	Automated GUI testing tools	30
4.5	Conclusion.....	33
5	Tested web application	34
6	Test automation	37
6.1	Test case repository	37
6.2	IntelliJ IDEA	38
6.3	Maven.....	39
6.4	Page object pattern.....	40
6.5	Writing automated test.....	44
6.6	Version control.....	45
6.7	Test execution and reporting	46

7	Discussion	49
7.1	Economic evaluation	49
8	Conclusion	51
9	Literature	52
A	Additional CD	55

1 Introduction and aim of this thesis

1.1 Introduction

Quality assurance is inseparable part of the product development cycle. There can be arguments about the necessity of the execution of automated test with every small change in the product. Some people could say, that the check of software functionality should be done only in crucial parts of the product development cycle. That would surely cause saving of time at the expense of quality. And in the modern dynamic society time is really the source, we are looking for. But product quality and stability are the attributes, which can never be reduced, neither in order to save the time. However, it would be really time consuming and ineffective to perform tests on software manually every time something has been changed. So what could be possible solution to execute continuous testing and save the time too? The answer is automation of testing.

Automation in general, not just in the connection with testing, is really crucial part of all the business processes. Automation is able to provide more stable, more efficient and more consistent results, like the results retrieved from doing the same activity over and over again. Additionally, it is better to develop system, which will do the same thing once, as the opposite of doing the same thing multiple times. Automation is very popular and every established company is trying to integrate it into its business processes.

The aim of this work is to take a closer look on test automation. In the thesis, there will be presented and analyzed tools and technologies, which can be used to automate software testing. The thesis will describe purpose and functionality of software product, which will be tested, as the part of this work. There will be presented a test suite, developed by Blackboard Czech s.r.o within the schoolwires quality assurance automation project.

1.2 Aim of this thesis

The aim of this thesis is to evaluate advantages of test automation and to compare test results depending on the used internet browser, depending on the time consumption and depending on the code changes. Output of this thesis should speed up product development process. It should contribute to providing new version of product to customers in the shortest possible time and should also increase the product quality.

2 Literature search

Software testing and test automation are very common terms in the field of product development process. There exists plenty of sources to obtain information from. One of the book which was used as the source of information for this thesis was written by Bill Laboon and it is pretty new.

A long time ago, in a company far, far away, I was a test lead. One of my responsibilities was to interview potential test engineers for our team, and I found that many candidates did not have a background in testing. Those who did well in their career often picked things up as they went. Even those who had degrees in computer science (or a related field) had often not learned about software testing. Developers had to learn how to properly test their own code, often following the old "apprentice" model by asking more senior developers what needed to be tested. (Laboon, 2016).

The book contains a lot of good information and it is very balanced between theory and practical examples. It is aimed at software testing in general but also describes test automation.

Another searched book is more focused on automated testing and contains advanced approaches which can be used to test a software. It is written by Frank Cohen and it is oriented on JAVA developers. The book provide information about both testing and development of JAVA based applications and based on this fact, it can be useful for developers also. It also has author's practical experiences and stories about testing and designing applications. (Cohen, 2004).

Software testing is nowadays popular topic and was also described in bachelor or diploma thesis. One of the bachelor theses written by Martin Bryndza contains good fundamentals about software testing and test automation. It also describes practical example of how can be test automation performed. It is very well written and can figure as good source of information for starting with automation of testing. (Bryndza, 2012).

There are also web articles which also provides some value and can be used as sources of knowledge about testing. The article written by Adam Kolawa is about regression testing and why is regression testing so important. Software development organizations with effective regression testing policies significantly improve the effectiveness of their software development staff and the success of their projects. Early identification of problems introduced by code modifications can save countless hours of development time spent chasing and resolving software errors, and allows the team to maintain and modify the application without fear of breaking previously-correct functionality. (Kolawa, 2016).

Another good book about testing software is written by Lee Copeland and can provide good knowledge for testers, test designers, software engineers etc. (Copeland, 2004).

Literature mentioned above and its containing information were used as the fundamentals for next chapter of this thesis.

3 Software testing

As Bill Laboon mentioned, testing software is a big part of the software development process, and useful not just for those looking specifically for a career in QA. A developer who doesn't care about software quality is not a good developer.

Quality of the software is very important aspect, especially before release of the product. This quality is supported by software testing. But what exactly is software testing? Is it finding every single bug or defect in the program? Is it the activity which is done after all the implementation is completed? Is it the process you will start after first complaints from unsatisfied customer? The answer is, it is none of those mentioned things. Before revealing of the definition of software testing, it would be appropriate to describe why software testing is so necessary.

It is very hard and some people would say it is impossible to measure the quality of software. It cannot be described by numbers, it cannot be given some value. But customers have to decide which product they will prefer the most. They can decide by functionality and complexity of the system, but these aspects are not the only ones that matters. There is also safety and stability. We can say that customers go through certain risk within the purchase of software. This risk can involve loss of data, failures of system or increased costs. It is the amount of risk which will be taken with the purchase of software, what is important. And finally, it is the software testing that provides estimates of how high is the risk customers will have to take on by purchasing software product. (Laboon, 2016).

3.1 Testing methods

Difference between **static and dynamic** testing is determined by the fact, whether the tested software has to be run or not. Static testing does not require program to run and it can be done before the very first release. There is also possibility to perform static testing even before start of writing the code. This can lead to better calculation of estimates for the future development. On the other hand, dynamic testing requires for its accomplishment existence of executable build and its behavior is measured by providing different entries and reviewing the outcomes. (Bryndza, 2012).

Software testing methods are traditionally divided into **black-box testing** and **white-box testing**. These approaches describes how the test cases will be designed. Between these two categories started up the third one, which is called grey-box testing.

In **white box testing** tester have full access to the source code of software and the test cases are designed according to it. Using this type of box testing tester not only sees what is happening outside of the application, but he also sees internal processes within the system interface. This scenario can lead into the lack of user experience testing, but it can be easier to determine which parts of code caused errors in the system. In many cases, white box testing can easily lead to testing implementation details instead of software use cases.

Within the **black box testing**, program is considered as a black box with described functionality. There is documentation of software functions, but internal codes are hidden from testers. In other words, it is very well known what the program should do, but the description how he achieves is not available.

Grey box testing uses principles of both mentioned types and combines them. In this type of testing tester does not have full access to the source codes of tested software, but he has information about basic principles of program's behavior. This description is based on thesis by Martin Bryndza. (Bryndza, 2012).

3.2 Testing levels

Unit testing is process of execution and validation of the system component in order to check its functionality. In the object oriented programming one unit represents most likely one class and minimum test case covers validation of constructor and destructor. Unit tests are usually written by developers during software development. Unit testing is designed to assess the units produced by the implementation phase and is the "lowest" level of testing. In some cases, such as when building general-purpose library modules, unit testing is done without knowledge of the encapsulating software application. As with module testing, most software development organizations make unit testing the responsibility of the programmer. (Ammann, 2008).

Integration testing covers verification of contract between single modules which already passed unit testing phase. These modules could be validated iteratively or all at the same time. The most important part of this testing is to validate communication between modules. There is a possibility that all of the modules would successfully pass unit tests and everything would seem to work right, but after connecting all of the modules into one system, there could be serious defects which could lead into whole system failure. In general integration testing's main task is to prevent such horrible scenario. Integration testing is described in book written by Bill Laboon. (Laboon, 2016).

System testing tests all of the modules which passed integration testing and the whole system itself integrated with any applicable hardware. By applicable hardware, it could mean desktop computer for testing desktop application or server for web application. System testing is designed to determine whether the assembled system meets its specifications. It assumes that the pieces work individually, and asks if the system works as a whole. This level of testing usually looks for design and specification problems. It is a very expensive place to find lower-level faults and is usually not done by the programmers, but by a separate testing team. (Ammann, 2008).

As the basic illustration of system testing example can be used this scenario:

1. Log in to the system
2. Create content
3. Validate content is created
4. Edit content
5. Delete content
6. Sign out from the system

The main function of **operational acceptance testing** is to determine whether product is ready to be deployed on production or not. This type of testing is usually performed by customers, which are able to use application in order to test its functionality. Acceptance testing is the final method of testing process and after its successful execution, there are no obstacles to release the system into production. (Laboon, 2016).

3.3 Types of testing

During a testing process, there exists a huge scales of types of tests which can be executed and every single one of them has its own purpose. In this subsection, all of the considerable types of test will be generally described. This section is inspired mostly by Bill Laboon's book. (Laboon, 2016).

Alpha testing is the very first stage of testing and it is performed in the early stage of testing. It is highly recommended only to test software by developers. After detecting and fixing basic software defects, testing process can go forward to beta testing. (Copeland, 2004).

Beta testing is preceded by alpha testing and is often called as the second phase of testing. It is possible to consider this type of testing as a form of user acceptance testing. It basically involves release of the beta version into external environment of the software development team. The software is available to the small group of people, usually future customers. Big contribution of beta testing is the amount of feedback which is possible to get. It is necessary to mention, that beta testing is mostly performed after the end of internal testing, which means testing by developers and quality assurance team is done. But this is not the rule. (Copeland, 2004).

Concurrent testing monitors the performance and provides outcome of the software under test during its normal activity. Concurrent testing is executed simultaneously with the functional testing and its goal is to determine stability and performance under expected circumstances.

Sometimes also called negative testing. **Stress testing** tries to make software fall. It tries to do so by providing dangerous test resources, such as very long texts,

unknown characters or scripts. The main purpose of stress testing is to test system recoverability which is very important aspect of robust software.

The purpose of **installation testing** is to determine whether software is installed correctly and it is fully functional on actual customer's hardware. This type of testing has its own opposite, which is uninstallation testing and its purpose is to check if the software was uninstalled correctly.

A common cause of failure of software is because it was tested only in one target environment. In this case, environment can be represented by web browser, operating system, maybe mobile platform. It is impossible that from all of the customers which are buying certain product, every single one of them is the same. They can have one thing in common which is the use of the same software product, but not all of them are using it on the same environment. And so as the every customer feels need to find the match with his desired environment, it is necessary to anticipate all of the customer's needs. This is supported by **compatibility testing**. (Copeland, 2004).

Regression testing identifies when code modifications cause previously-working functionality to regress, or fail, ultimately allowing you to catch regression errors as soon as they are introduced. Most organizations verify critical functionality once, and then assume it continues to work unless they intentionally modify it. However, even routine and minor code changes can have unexpected side effects that might break previously-verified functionality. (Kolawa, 2016).

This quote point out on big demand for integration of regression tests into testing process. Although, these tests are very complex, large scale and have to be executed continuously. Because of these factors, regression tests seems to be a best candidate to being automated. It may seem as an easy task, but for the purpose of future consistency and stability of regression tests, there have to be established a strong policy for writing the automated test cases. Otherwise, technical debt of the code would grow larger and larger until the whole project could fall apart.

A smoke test involves a minimal amount of testing that can be done to ensure that the system under test is ready for further testing. It can be thought of as a guard to further testing, unless the system can perform some minimal operations, you don't move on to running a full test suite. This is usually a small number of tests, checking that the software can be installed, that major functionality seems to work, and that no obvious problems appear. (Laboon, 2016).

There exists various types of tests, which can be used and every type provides information about another aspect of software. It is very important to take care of execution of as many tests, as it can be done in available time capacity.

3.4 Testing process

There exists a huge variety of methodologies which are available to use. The usage of right approach is basic concept within the software development. There exists possibility, that previously chosen approach does not fit for the software development in the future and that another methodology could lead into more efficient

work. Substitution between different methodologies can be solution for this case, but this substitution can lead to additional costs in form of time, money etc.

Methodology represents model of policies and responsibilities of every member of the software development group. Every member should be familiarized with his responsibilities and should be adjusting his or her working behavior to them. Some of the basic methodologies will be described below.

3.4.1 Waterfall development model

The oldest model of development cycle. Its name comes from comparison of sequence of particular phases and waterfall water flow. The basic concept of this model is based on sequence approach to each of the phases. Based on the model, prerequisite to start a new phase is that previous phase have to be completed successfully. The early phases of development have to be executed very carefully and closely, because revealing of a program defect in early phase have less impact on time consumption, than it would have in the future phases.

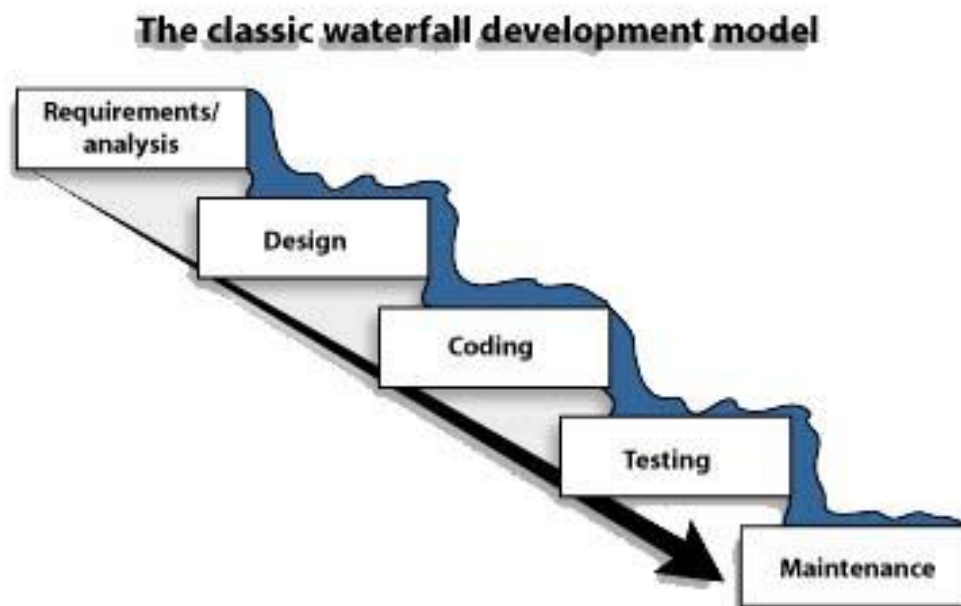


Figure 1 Waterfall development model
Source: Railshorde, 2016

During development of huge projects, there will always be need to return to the previously completed phases. It can be caused by change of customer requirements. The biggest disadvantage of waterfall model is that return to the previously completed phases is impossible, so adjusting to the customer changing requirements can be difficult, using this development model.

Rational unified process (RUP) is based on waterfall model and has better ability to adapt to changing customer requirements. RUP consists of iterations and

every single iteration consists of all phases mentioned in waterfall development model. Basic concept of rational unified process will be described below.

In each single iteration, sequence of individual tasks and its time estimate is set. The best possible situation happens when all tasks of iteration are completed. Most of the time in early iterations is spent by meeting and analyzing customer's requirements. After analyze, function specification is created. This specification consists of new and existing functionality. Based on function specification, development process which includes also testing process, can start. When all the iterations are completed successfully, one additional test of all iterations is performed. After test of all iterations, software is ready to go through acceptance testing, which is necessary before deploying new release to production. This section was based on book written by Ivana Rábová. All detailed information about waterfall development model or RUP methodology is described in the book. (Rábová, 2008).

3.4.2 Agile development model

Agile development is popular. All the cool kids are doing it: Google, Yahoo, Symantec, Microsoft, and the list goes on. I know of one company that has already changed its name to Agili-something in order to ride the bandwagon. This quote is taken from book written by James Shore and Shane Warden. Content of this section is based on the book as well. (Warden and Shore, 2008). Nowadays, plenty of companies integrating agile methodologies into their business model. However, it is necessary to specify how it will be done. For the best performance of this methodology, client have to cooperate during all development process.

Agile methodology is the process of planning and verifying work. The aim of methodology is to improve organization of work. The whole development process can be distinguished to this phases:

1. First iteration – First analysis and implementation of basic functionality.
2. Analysis of change – Select what will be implemented
3. Implementation
4. Presentation to the client
5. If the product is not finished back to step 2
6. If the product is finished, maintenance and improvements takes place

Scrum is one of the most known agile methodologies and can be applied on the team of four to fifteen people. There are two groups of people performing in scrum. Group which is committed and group which is involved. Committed group consists of product owner and scrum master. Product owner is responsible for what will be implemented in the upcoming sprint and specifies implementation details. Aim of the scrum master is to manage developers. He should satisfy their needs, solve their problems etc. The other group consists of stakeholders and managers.

Development phase in sprint is called sprint. Sprint has its time duration and it is often 14 days. Before start of new sprint, there is always sprint planning. In sprint planning, all of the user stories are listed and then moved to product backlog. Within the sprint planning, user stories are created, they are associated with estimates and it should be discussed which of the stories will be implemented in the upcoming sprint. After the end of each sprint, whole team has retrospective meeting, where anybody can bring their suggestions or complains.

With start of sprint comes also start of development, which covers implementation of user stories. After story is implemented, it goes to code review. If other developers approve the review, user story can be marked as done and implementation of another story can begin. Each day, whole team attends stand up, where each member of the team describes scope of his work.

Scrum is based on good communication within the whole team, which leads to better work efficiency. All member of the team should cooperate and help each other to reach the goal faster.

4 Test automation tools

Tests can be marked as automated, if it is possible to run test cases without an interaction from the side of tester. Automated tests should meet these conditions (based on Michal Gajdošík):

- Ability to execute set or subset of test cases
- No need for interference after the start of tests
- Important test parameters are set automatically
- Test results are registered automatically
- Ability to compare actual test results with expected results
- Capability to analyze test results and provide outcome

4.1 Advantages of test automation

Not all of the contributions of automated testing are visible from the beginning of project. Some of them can be unexpected. The other can be seen not as a result of automation. The benefits of test automation can be discovered in future stages of project or just after development of tests is done and project is in maintenance phase. The main benefits of test automation are listed below. Douglas Hoffman in his document mentioned cost and benefits of test automation. Some of the benefits are mentioned below. Content is inspired by Douglas Hoffman. (Hoffman, 2016).

Price is the factor that every company should consider. Automated tests can run independently, which leads to saving of human resources. It also provides saving of hardware resources, because all of the tests can run on one single machine. In the development phase, implementing tests can seem like a waste of time, but in the future, costs invested to development of tests will make huge impact to savings of time, people and hardware.

Reusability is one of the main advantages of automated testing. Do one thing repeatedly can be ineffective and in most cases, it can become boring which leads into loss of focus. The idea of complete a task once and never do it again, seems like a best option. This sounds nice, but to write stable test, there are more things to do just a stable code. Firstly, software analyze have to be done. The person who are writing test should be very familiar with the tested software. Then, all the expected outcomes should be analyzed and test codes should be adjusted to them. In-depth analyze provides good fundamentals for implementing stable test cases.

Speed of the test execution is also important. After integration of test automation, regression testing will take less time as the manual regression testing. However, not all the test cases can be automated, but on the other hand, manual testing can be executed at the same time as automated test, which will lead to save of time.

Correctness is one of the important factors too. Each test will do same steps every time. Manual testing can't provide this accuracy because it is performed by humans and as we know, every human is fallible. With the huge amount of tests to

execute and combined with exhaustion, it is possible that some steps of test will be overlooked and test will not be executed properly.

Test automation comes with many advantages. Some of them are smaller, some of them are more obvious. But it also needs sacrifice in early stages of project. And if these sacrifices come through, there is deserved reward in the end.

4.2 Unit testing frameworks

Unit testing frameworks are usually not part of the compiler suite. These can be understood as a third party products which contributes to testing process. Some examples of testing frameworks available on the market will be described below.

NUnit is open source framework supporting all .NET languages. This framework allows users to create and run automated unit tests. In order to develop automated tests using NUnit, framework and text editor are required. Although, text editor can be replaced by integrated development environment, for more convenient work. To demonstrate basic principles of NUnit, simple test case will be written and described in Figure 2, using NUnit framework and Monodevelop IDE.

```
1 using System;
2 using NUnit.Framework;
3 namespace SimpleTestCase
4 {
5     [TestFixture]
6     public class TestExample
7     {
8         [Test]
9         public void positiveTest() {
10             int x = 5;
11             int y = 5;
12             Assert.AreEqual (x,y);
13         }
14
15         [Test]
16         public void negativeTest() {
17             String shortText = "text";
18             String longText = "This is long text";
19             Assert.AreNotEqual (shortText.Length, longText.Length);
20         }
21
22         [Test, ExpectedException(typeof (NotSupportedException))]
23         public void testException() {
24             throw new NotSupportedException ();
25         }
26
27         [Test, Ignore]
28         public void testNotImplemented() {
29             throw new NotImplementedException ();
30         }
31     }
32 }
33
```

Figure 2 NUnit test case example

Each class that contains test methods have to be annotated with statement `TestFixture`. This tells framework where to find testing methods. Each one of test methods is annotated with “Test” statement, which indicates that framework will execute this method. `Test` annotation can be filled with additional properties, like it is described on the picture above. For example, `ExpectedException` tells to catch exception of given type. It is worth mentioning that in other cases, exceptions cause failures of the test. `Ignore` statement can be used to ignore the test method and don’t execute it at all. This feature can be used, when test procedure is not implemented yet, or is not implemented at all. It is just defined to describe structure of the test class. In the picture, there are also very simple examples of positive and negative testing.

Execution of tests, using this framework can be done by NUnit agent which is available within the framework directory. Framework’s compressed package can be downloaded at link: www.nunit.org. In agent, user is able to simply choose classes which he want to test and run the tests. Nevertheless, this is not only option. `Monodevelop` provides more simplified and convenient method to run tests, within the IDE. There is NUnit test plugin available for free, which can run and provide test results directly from development environment. Output from the test cases

written in Figure 2 is available in Figure 3. From the picture, it is obvious that ignored test was not executed.

After review and analyze of NUnit framework, it seems to be a very good solution to use for testing .NET based projects. With combination with Monodevelop plugin, it can act like reliable tool for writing and executing unit test, but also as a stable testing tool for continuous development.

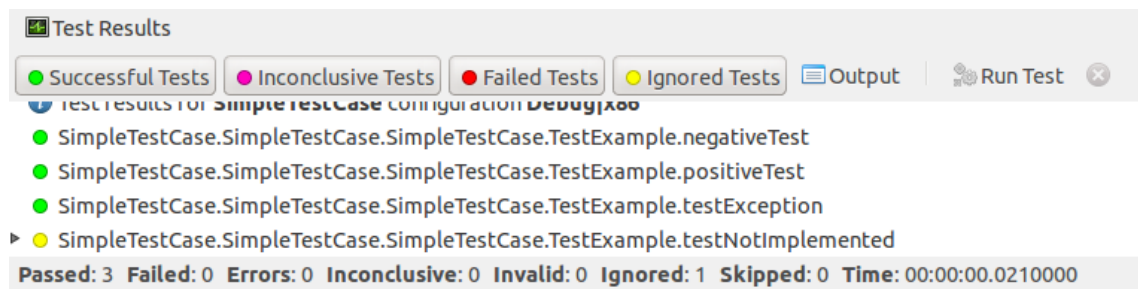


Figure 3 NUnit test result example

RSpec is the most popular ruby based testing framework and has a huge community of developers behind it. As was mentioned, RSpec runs on ruby, but framework has plenty of its own add-ons on the top of ruby programming language. You could say that RSpec is what is traditionally known as a unit testing framework, but we prefer to describe it as a Domain Specific Language for describing the expected behavior of a system with executable examples. (Chelimsky, 2016).

Before start writing tests with RSpec, Ruby 1.8.4 or later have to be installed and of course, RSpec gem have to be installed. These prerequisites can be done by typing following commands to terminal (These commands were tested on Linux based operating system). `Apt-get install ruby` and `Gem install rspec`. After these two commands are executed successfully, prerequisites for starting test automation with RSpec are done.

In RSpec, there are some little naming changes, compared to another test frameworks. Statements like `test case` and `test method` are not used at all. Instead of them, RSpec uses terms `behavior` and `example`. It could be little bit confusing, because on the image image below, there are non of these terms used. To describe behaviour of tested class, there have to be used keyword `describe`. Remaining `examples` will be instantiated using `command it`. It is required to mention, that during development of any kind of an application, using RSpec, developers should first implement `behavior` and after that implement remaining functionality. Examples of behaviour and its output is presented in Figure 4.

RSpec performs very well and for ruby developers, it should be friendly tool. Installaton of Rspec is simple, even writing simple test cases. Test results are readable and finding of an error should not be a problem in this case. A little disadvantage of Rspec is the lack of complex available IDEs in the market. A lot of free IDEs are old and the new ones are always commercial.

```
1 describe User do
2   it "should be in any roles assigned to it" do
3     user = User.new
4     user.assign_role("assigned role")
5     user.should be_in_role("assigned role")
6   end
7
8   it "should NOT be in any roles not assigned to it" do
9     user = User.new
10    user.should_not be_in_role("unassigned role")
11  end
12 end
```

```
$ spec user_spec.rb --format specdoc

User
- should be in any roles assigned to it
- should NOT be in any roles not assigned to it (FAILED - 1)

1)
'User should NOT be in any roles not assigned to it' FAILED
expected in_role?("unassigned role") to return false, got true
./user_spec.rb:12:

Finished in 0.019014 seconds

2 examples, 1 failure
```

Figure 4 RSpec behaviour and result example
Source: Chelimsky, 2016

PHPUnit provides framework to write and execute unit automated tests in PHP programming language. Framework is available for free and it is very popular within the PHP developers. However, framework seems to have use only as a unit testing tool for PHP based projects. To implement test case using this framework, user have to have it downloaded. If so, with simple extension of framework class, test case can be implemented. Example of unit test is displayed in Figure 5. This test just validates correct functionality of the constructor class.


```
<?php
class CarTest extends PHPUnit_Framework_TestCase
{
    public function carCreatedTest()
    {
        // Arrange
        $car = new Car(4, 'Black');

        // Assert
        $this->assertEquals('Black', $car->getColor());
        $this->assertEquals(4, $car->getWheelsCount());
    }
}
```

Figure 5 PHPUnit test case example

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc. (TestNG). Requirements for this framework is java development kit 7 or higher. TestNG comes with plugins for IDEs such as Eclipse, NetBeans or IDEA IntelliJ. In this particular case, TestNG framework with integration for IntelliJ IDEA will be demonstrated.

There are plenty of annotations which can be used to develop large scale test cases. The most common are `Test`, `BeforeClass`, `AfterClass`, `BeforeMethod` and `AfterMethod`. `Test` annotation simply tells that this is a test case. `Before` and `after` methods are executed before and after each test case. These methods can be used as some kind of setup, or clean up for the test method. `BeforeClass` method is executed before the very first command of the test class. On the other hand, `AfterClass` method is called when all of the other lines of the code from the test class are finished.

Test execution within the framework consists of implementation of test cases, annotating them and executing the xml configuration file. Class containing tests can be annotated by `Test` annotation and all the public methods will be handled as test methods. Simple example of test is described on images below. On the left side of image, there is example of test class written in pseudocode. `Groups` parameter tells that test belongs to group. Using this feature, tests can be separated into the groups, based on type of testing, subject of testing etc. On the right side, there is a XML configuration file which specifies test run configuration. Between `classes` tags, there are all test classes which will be executed and between `run` tags using `include` tag, it can be specified which groups to run. Execution can be done within the mentioned IDEs.

TestNG provides a lot of advantages to improve test development and execution. Based on that, this framework will be used and demonstrated in practical part of this thesis.

```

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

@Test(groups = "example-tests")
public class TestNgExample {

    @BeforeMethod
    public void signIn() {
        // Sign in user
    }

    public void validateSignIn() {
        // Check if user is signed in
    }

    @AfterMethod
    public void signOut() {
        // Sign out user
    }
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<suite name="ExampleTestSuite">
  <test-name>
    <classes>
      <class name="TestNgExample"/>
    </classes>
    <groups>
      <run>
        <include name="example-tests"/>
      </run>
    </groups>
  </test-name>
</suite>

```

Figure 6 TestNG example

4.3 Automated web testing tools

With the growing number of web-based applications, there is growing need for automation of web applications. There are plenty tools which can be used for automation. Some of them comes as web browser extensions, some of them are commercial desktop applications. Automation of web application can be challenging in some cases, because changes in user interface can be significant. These changes are mostly caused by usage of different browser that was used for developing the tests. However, using good approach and right tools, these obstacles can be solved. Some examples of automated web testing tools will be described below.

Selenium comes with two solutions for test automation. As first will be mentioned Selenium IDE. Selenium IDE is a free Mozilla Firefox extension. It enables users to record, debug and edit simple automated tests for web-based applications. On the left side of IDE, there is the list of all test cases. These test cases can be run one by one or as a whole suite. Also, there is a possibility to set up tests to run periodically. Test cases can be recorded, or wrote manually. Previously recorded cases can be edited within the IDE, by changing steps of the test case. Test steps are displayed in the middle of IDE and are executed from top to bottom. User can also switch view between test steps from normal to source. This source code is generated as html code.

Advantage of Selenium IDE is its simplicity of creating tests. On the other hand, it is not convenient for writing robust test cases. Also, test generated by this IDE could be very hard to maintain in the future.

Selenium also comes with its web driver form. It provides set of frameworks for different programming languages. It also supports various internet browsers, but behavior of the test based on chosen browser can be different. Web driver also provides `HtmlUnitDriver` which executes tests without rendering browser window. Before every test, web driver starts browser, creates user profile and installs driver. Within the web driver, tester can go directly to given URL, click on elements, waiting for elements, send text to input fields etc. It also can call JavaScript functions and switching between more browser windows.

Selenium web driver is very strong tool for writing automated tests and with its large functionality, it can handle a lot of challenges which can appear during test development. This tool will be used and demonstrated on practical examples, later in the thesis. Test example available in Figure 7 demonstrate Selenium IDE workspace.

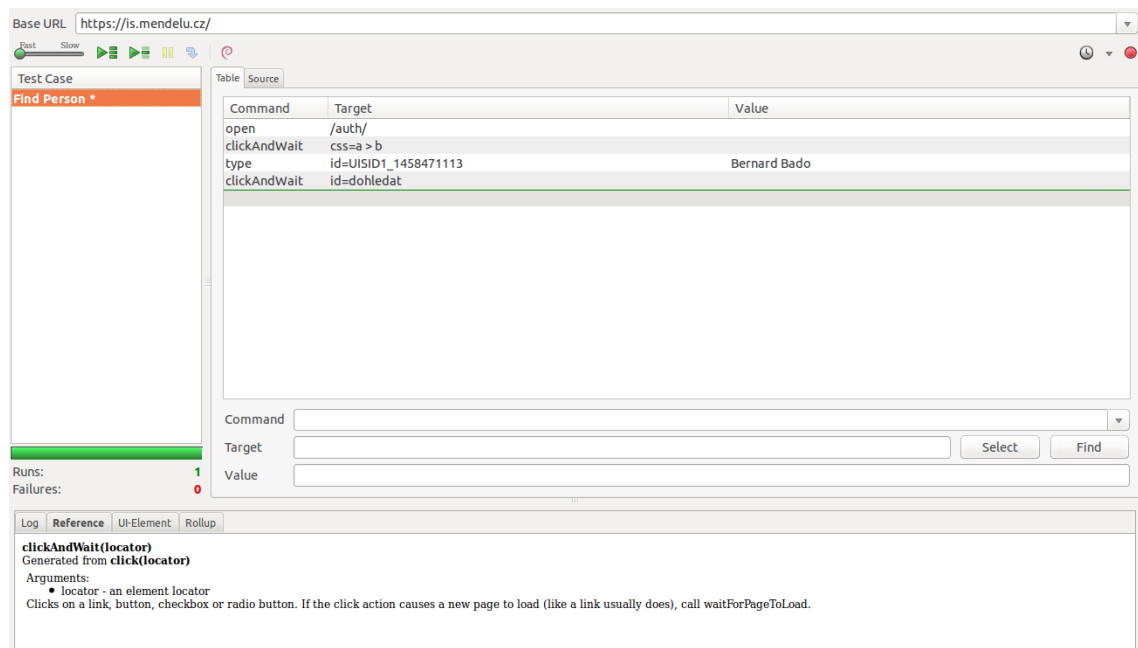


Figure 7 Test example in Selenium IDE

SoapUI is a free and open source cross-platform Functional Testing solution. With an easy-to-use graphical interface, and enterprise-class features, SoapUI allows you to easily and rapidly create and execute automated functional, regression, compliance, and load tests. In a single test environment, SoapUI provides complete test coverage and supports all the standard protocols and technologies. (Smartbear, 2016).

Test structure in SoapUI is similar like in another mentioned tools. Test suite contains test classes and these classes contain test steps. Basic test step is repre-

sented by sending request to web service. Another test steps can be database requests. Database provides data which can be used as the source of next test step. Test steps within the test case are executed from top to bottom. Tool supports Assertions to verify correct execution of test steps.

SoapUI seems like comfortable tool for executing functional and load tests, but its usability within the regression testing could have more disadvantages than advantages. Tools does not have to be used as standalone program, there are also plugins available for various IDEs like NetBeans or Eclipse.

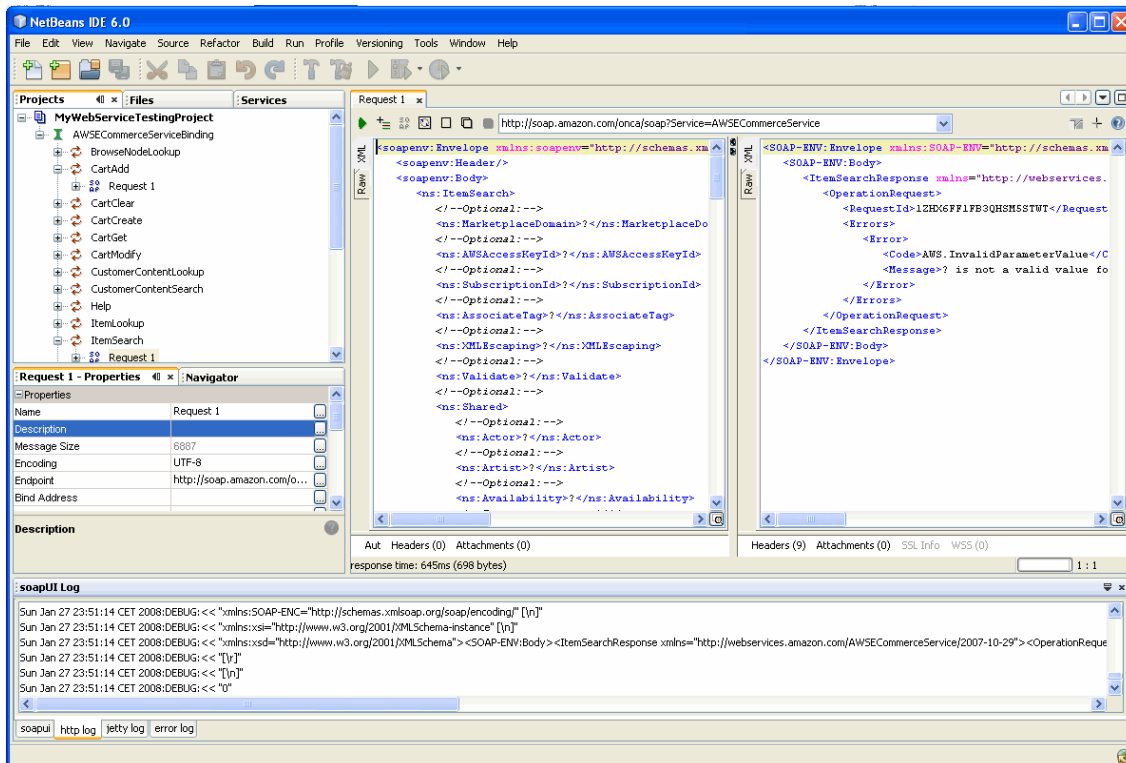


Figure 8 SoapUI plugin for NetBeans IDE

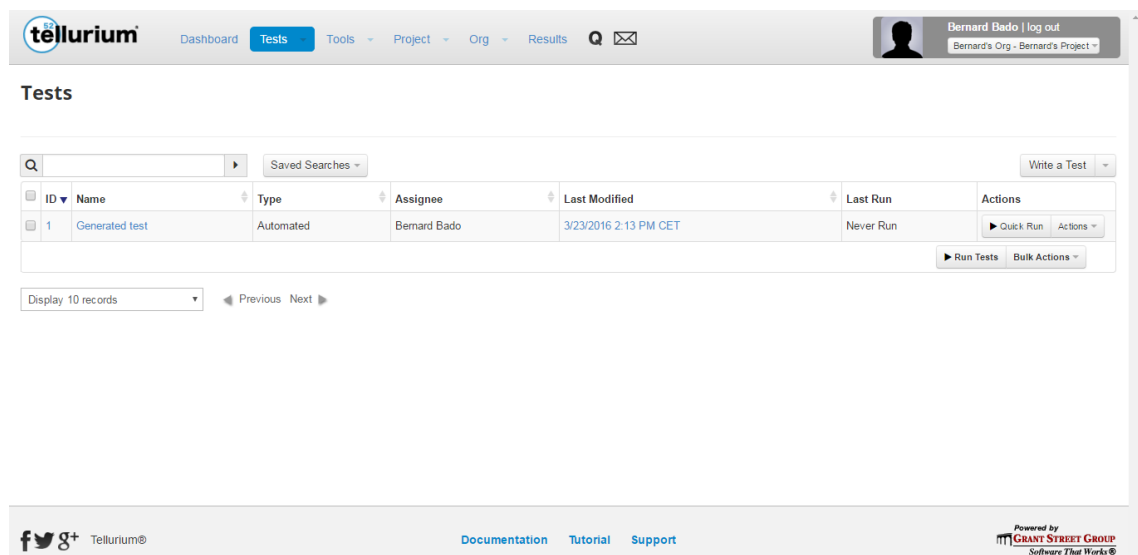
As another testing tools will be mentioned **Tellurium**. The biggest advantages of tellurium is that for automation, all the tester need is internet connection and internet browser. Tellurium is web-based application providing ability to write, generate, execute and maintain tests for web applications. All created tests are available within the application. Tests can be formed to groups and executed separately. Another positive thing about tellurium is that for writing automation tests, there is no need for knowledge of any programming language. Test cases in this tool are described by structured English, which leads to better readability of test cases.

For the url <http://www.xe.com/currencyconverter/>
Then the "The World's Trusted Currency Authority" link is present

For imagination, example code above opens given URL and validate that the link `The World's Trusted Currency Authority` is visible on the given page. Tellurium provides examples of commands, available for usage within the test case. Tellurium also provides ability to generate test case for the given webpage. It is able to generate test case just for one web page, not the whole web content. It is important to mention that these generated tests only validate structure of page. In order to create more complex test cases, tester have to start from scratch and write tests using structured English.

Example of tellurium application can be seen in Figure 9 **Error! Reference source not found.** Within the top panel, there are options for managing tests, executing tests, seeing test results and another additional configurations. In the center of application, there are all saved test cases. These test cases can be search and filtered by name, group, type etc. Below list of all the tests, there is button for execution of tests. After execution is done, results can be seen in tab with the corresponding name.

Writing automated tests in Tellurium seems such an easy task. Tester don't need a lot of knowledge, all available commands are available within the application helper. Each command contains example of its usage. It all can seem, tellurium is ideal choice for test automation. But this tool is convenient just for simple use cases and does not contain functionality which is needed to write more stable and more robust tests. Another important thing to mention is that tellurium only execute tests in its built-in web driver, so it's impossible to compare test results, using different web-browser. However, tellurium can be good tool to start with and get some experience with testing of web applications.



The screenshot shows the Tellurium application interface. At the top, there is a navigation bar with the Tellurium logo, a 'Dashboard' tab, and a 'Tests' tab. The 'Tests' tab is active, showing a search bar, a 'Saved Searches' dropdown, and a 'Write a Test' button. Below this is a table of test cases with the following columns: ID, Name, Type, Assignee, Last Modified, Last Run, and Actions. The table contains one row with ID 1, Name 'Generated test', Type 'Automated', Assignee 'Bernard Bado', Last Modified '3/23/2016 2:13 PM CET', and Last Run 'Never Run'. Below the table, there is a 'Display 10 records' dropdown and 'Previous' and 'Next' navigation buttons. At the bottom of the interface, there are social media icons for Facebook, Twitter, and Google+, the Tellurium logo, and links for 'Documentation', 'Tutorial', and 'Support'. On the right side, there is a 'Powered by GRANT STREET GROUP Software That Works' logo.

ID	Name	Type	Assignee	Last Modified	Last Run	Actions
1	Generated test	Automated	Bernard Bado	3/23/2016 2:13 PM CET	Never Run	Quick Run Actions

Figure 9 Tellurium application

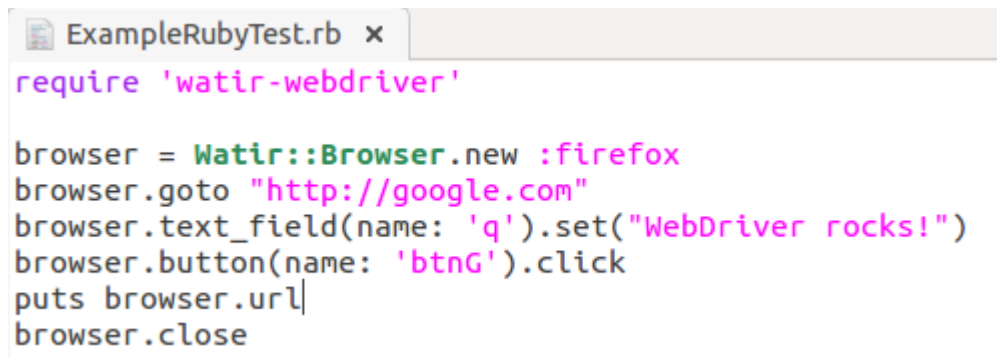
Watir is an open-source family of Ruby libraries for automating web-browser tests. This tools uses Ruby syntax with combination of its own features to create

complex test cases. Watir provides a lot of utilities for handling web browser interactions e.g. JavaScript dialogs, browser downloads, cookies etc. List of all supported interactions is available at Watir web driver site: <http://watirwebdriver.com/#in-Browser-Elements-and-Advanced-Interactions-section>.

Watir runs on Ruby, so it is probably obvious that Ruby have to be installed in order to use Watir. There are plenty of tools which can be combined with Watir, but in this particular example, only plain usage of Watir will be described. Ruby installing packages can be downloaded at: <https://www.ruby-lang.org/en/downloads/>. Users using Unix-like operating systems can download ruby with `sudo apt-get install ruby-full` command. With following command `gem install watir-webdriver` is everything ready for start of test development using Watir. It is important to mention that this tool is very similar to already mentioned Selenium web driver and it is not just based on likeness in names. Fact is that within the installation of Watir, Selenium web driver is installed also. It doesn't mean it have to be use, it is just part of the Watir package.

In Figure 10, there is very simple example of test using Watir we driver. This test initiate new Firefox web driver, opens URL on site <http://google.com>, set input of text search to `Webdriver rocks`, clicks on search button, get the output URL and close the driver. This example is very simple and should only demonstrate syntax of Watir. The syntax is clear and its readability is also convenient.

Watir seems like a very good choice for automation of web-based applications for Ruby based developers. It supports page object pattern and has a whole documentation about it available at: <https://github.com/watir/watir-webdriver/wiki/Page-Objects>. Page object pattern based on selenium will be described later in the thesis, within the practical part.

A screenshot of a code editor window titled "ExampleRubyTest.rb". The code is written in Ruby and uses the Watir library for browser automation. The code starts with a require statement for 'watir-webdriver'. It then creates a new Watir browser instance for Firefox, navigates to 'http://google.com', sets the text of a search field with name 'q' to 'WebDriver rocks!', clicks a button with name 'btnG', prints the browser's URL, and finally closes the browser.

```
require 'watir-webdriver'

browser = Watir::Browser.new :firefox
browser.goto "http://google.com"
browser.text_field(name: 'q').set("WebDriver rocks!")
browser.button(name: 'btnG').click
puts browser.url
browser.close
```

Figure 10 Example Watir test

4.4 Automated GUI testing tools

Testing of desktop applications is not very different from testing of web based applications. Desktop application have its own user interface like web application. Major difference is that these application don't run in web browsers, but within the operating system. Based on that, we could say, that automated tests for desktop

applications should cover huge variety of operating system. Of course, it only can be done if the tested application is supported by operating system. There are many tools available on the market and the more popular ones are for commercial use. Some examples of automated GUI testing tools will be described below. It is important to mention that tools which will be described can support automation of web applications too.

Test Complete is commercial software which provides support for creating automated tests for mobile, web and desktop applications. Within this framework, tester can create automated tests using script languages, using record of user interaction or with combination of the two. Test Complete supports script languages like C++Script, CScript, Jscript etc. There is also possibility to integrate Test Complete with Selenium or SoapUI. It also can be integrated with bug reporting tools so the process of finding possible bugs or defects can be minimalized, using this feature. Test Complete is commercial, however it is provided for 30 days free trial.

Within the creation of automated tests, there is **Test Visualizer** which translate one line of test code into its graphical representation. Say that in test code, there is command to click on button **Done**, visualizer will display it like button **Done** being clicked. Whole documentation, distinguished by operating systems is available at: <https://support.smartbear.com/viewarticle/75299/?st=0>.

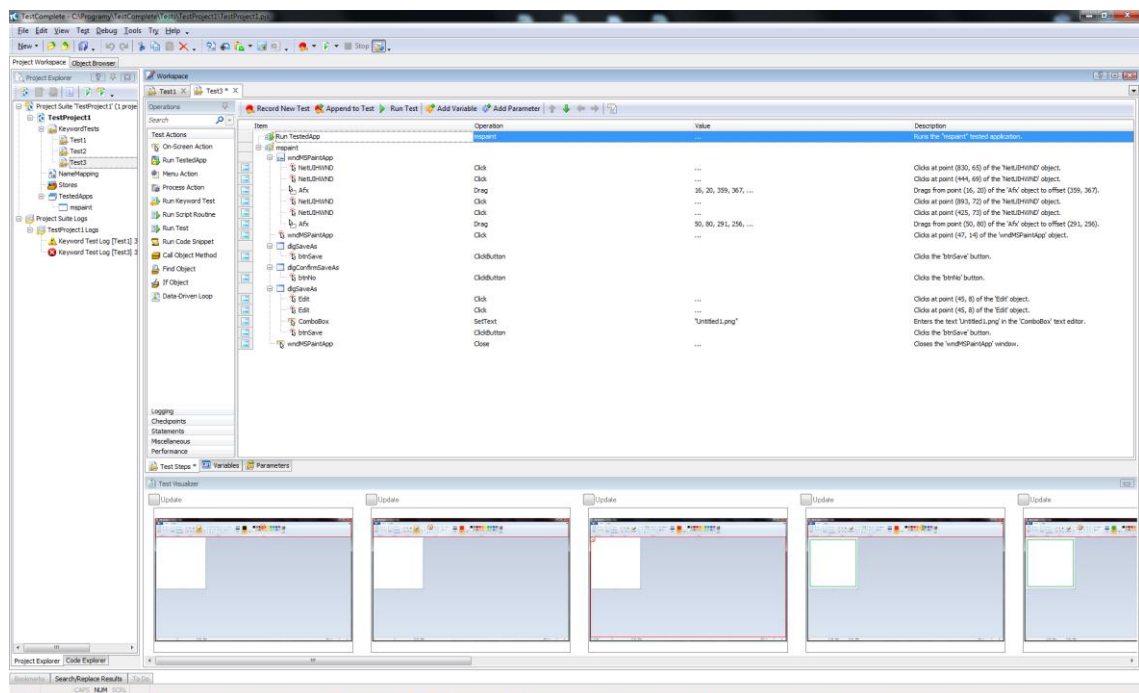


Figure 11 TestComplete workspace example

Ranorex Studio offers easy-to-use test automation tools for creating reliable automated testing projects. (Ranorex, 2016). Ranorex is easy to install, everything what is necessary for starting with test automation using this tool is contained in

one installation package. Installation package is available at Ranorex official website: <http://www.ranorex.com/>. Ranorex is commercial product and can be tested in free trial.

Test cases in Ranorex can be created by recording. When recording is finished, tests steps are saved within the test case. Tool also automatically process test steps and generate code from it. Code can be written in programming languages such as C# or languages based on .NET framework. Ranorex can be used for testing of desktop, mobile and also web applications. It basically can test all applications with graphical user interface. Ranorex can be also used as an IDE for writing automated tests without use of recording feature. Tool provides code completion which makes coding better and faster. But it would be good-for-nothing to use this tool just for coding. Test execution can be scheduled within the tool and test report is generated and updated simultaneously with execution.

Ranorex provides plenty of features and it is obvious that application is designed to support large scale projects. It has a lot of advantages that will make the automation easier, but as it is with the test recorders, it is hard to maintain the project as it grows.

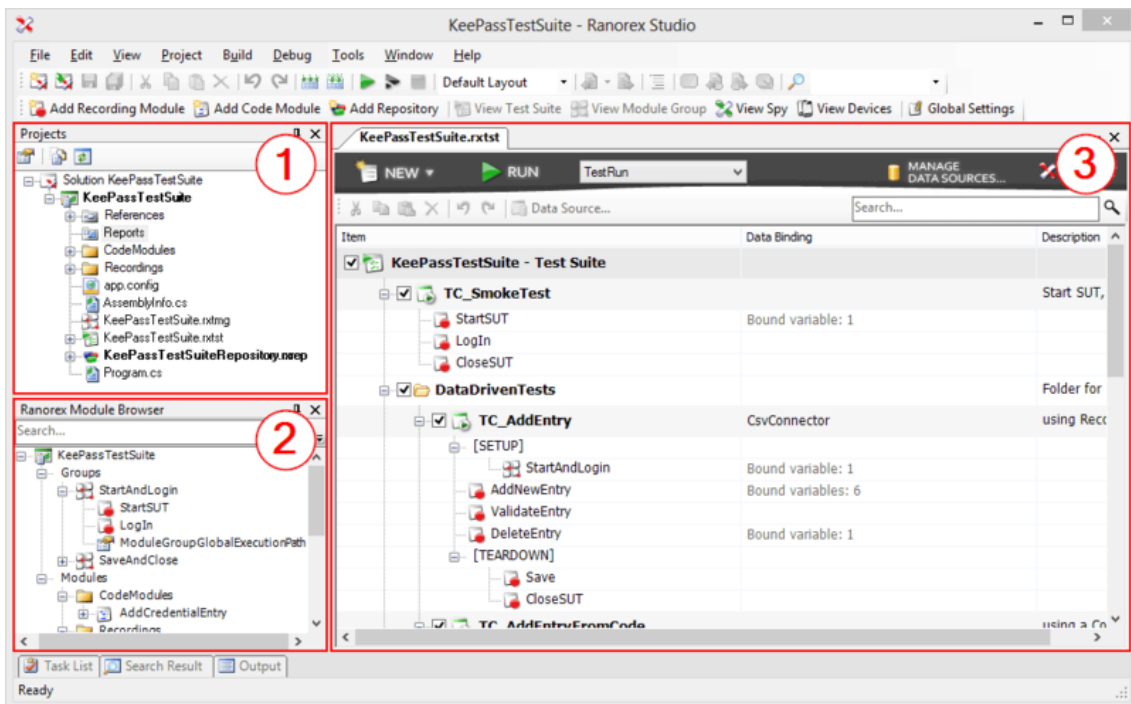


Figure 12 Ranorex studio layout
Source: Ranorex, 2016

Figure 13

4.5 Conclusion

In this chapter, tools available for test automation were described. Now we can see that there exists plenty of tools available. Some of them requires more programming knowledge, some of them are made for less experienced users. We can tell everyone can meet his requirements.

In this thesis, aim is to develop test suite of regression tests which can be executed on daily basis. As it was mentioned, regression tests usually simulate user interactions with the tested software. This simulation will be performed by selenium web driver. For test management, TestNG was chosen as the best alternative. Test cases and whole test project will be written in JAVA language. With combination of these tools, we can proceed to test automation.

5 Tested web application

In every case, there is a need to study and analyze tested application. Each software can offer different functionality and expected behavior. Based on lack of understanding how tested application work, it can happen that test development will consume much more time. It can be sometimes difficult task to explore whole software functionality. In some cases, it's not even required. Sometimes, maybe key features needs to be known and other not that important ones can be explored later.

Name of the software which will be tested is Centricity2. In this section, key concept and purpose of Centricity2 will be described. Purpose of the application is to help schools to create and maintain content management system in order to increase the actualities about the school itself. In other words, it helps school to create online presentation of the school. All content which will be rendered on end user page is created within the site manager page. Site manager has limited access and only trusted users can get rights to do changes in it. Site manager provides a lot of options to do, but main focus will be given on creating page content.

Page content consist of functional modules called apps. Each app has its own features and purpose of each app is different. Basic concept consists of creating a page by client's need by building it with provided apps. It is possible to choose from different page layouts. By choosing specific layout, placements of the apps can change and will change on the page. Example of managing apps and layout in site manager can be seen on Figure 14.



Figure 14 Management of apps and layout within the Centricity2

It is the right functionality of each app which is important to work. Based on this fact, tests can be differenced by apps and structure of test repository can reflect it as well. It is obvious from Figure 14 that this page contains one app with name Content App. Detail view of Content App in site manager page can be seen on Figure 15. This specific app is basically WYSIWYG editor with various options provid-

ed for content creators. In picture below, one heading and one picture were added to the content.

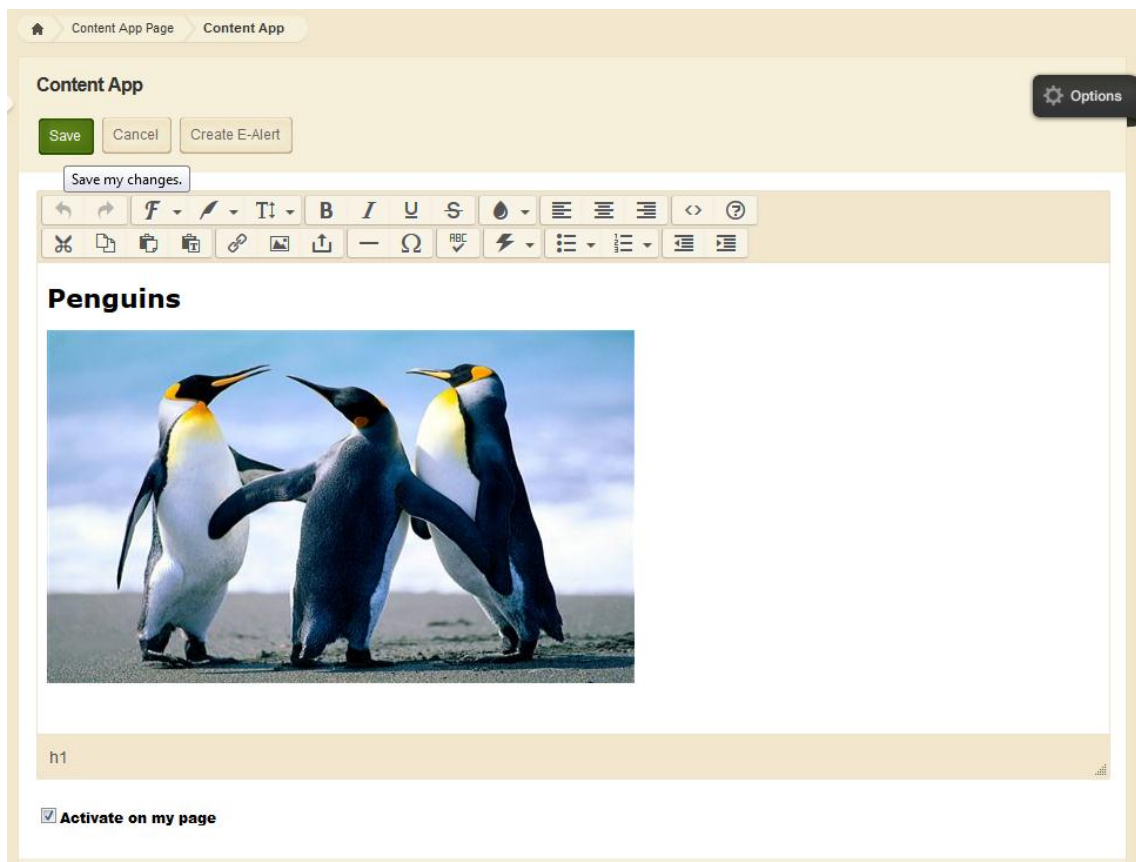


Figure 15 Detail of Content App in site manager page

Representing content from page on Figure 14 rendered on end user page can be seen on Figure 16.



Figure 16 Rendered content on end user page

6 Test automation

In this chapter, whole process of test automation will be described on practical example. As the main resource will be in service test automation project, developed within Blackboard Czech s.r.o. Project is developed and maintain by quality assurance team which also author of this thesis belongs to. In this chapter, some of already mentioned tools will be described more specifically so readers of this thesis could have better illustration of how the whole automation process goes.

6.1 Test case repository

It is always better to store important information which can be needed at more places at once somewhere where it can actually be at more places at once. For this purpose, cloud services may seem as a good solution. It can be true, but there are many tools for managing test cases available on the market. And these tools have bigger functionality like just storing some information. They comes with features like providing test run reports, given structure of test cases, managing structure of repository by given order etc. In this particular case, TestRail will be described.

TestRail helps you manage and track your software testing efforts and organize your QA department. Its intuitive web-based user interface makes it easy to create test cases, manage test runs and coordinate your entire testing process. (Gurock, 2016).

In the example of test automation process, following test case will be provided and process of its automation will be described as it goes. As the starting point, we will consider that test case is already written and its automation needs to be done. Example of test case in TestRail working directory is available on Figure 17.

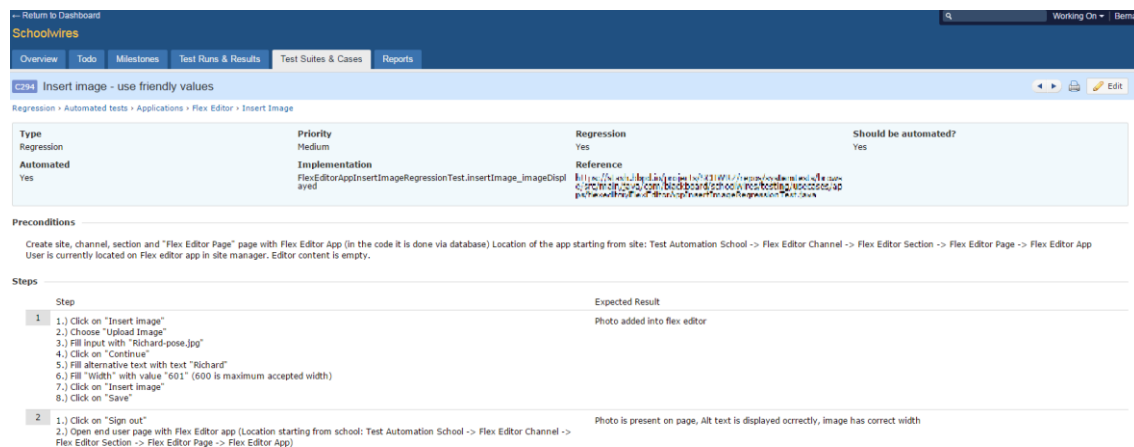


Figure 17 Example test case in TestRail

TestRail provides more comfortable way to write test cases. When tester is doing so, he basically just filling the form. When test case is finished, automation engi-

neer can be notified by TODO feature. It signalize that he has test case to automate¹. At this moment, test analyst can start work on different subject and process continues to automation engineer. He has to make previously created test case automated.

This means the process continues to next phase which is writing automated test. Before processing directly to next step, some useful technologies and patterns will be described. With use of these technologies writing single automated test is a very easy task. Technologies which can be used to do so were mentioned before. Reader should be at this point familiarized with technologies for automation, available at market. In our example, combination of several tools is used. For writing the code, it is the best option to use integrated development environment, because it provides nice features to make the work of programmer easier. In next subsection, attention will be given to IDE used within the Blackboard's project.

6.2 IntelliJ IDEA

IntelliJ IDEA (for future only IntelliJ) is product developed and maintained by company Jet Brains. Jet Brains comes with plenty of IDEs for various programming languages. The list of all available products is available at: <https://www.jetbrains.com/>. Licenses for the products are for each one different. About IntelliJ, it comes with two variants. The first one is community and the second one is called Ultimate. Community edition contains a lot of features and for the purpose of this thesis, it is adequate one. It is good to know that Jet Brains provides student partners program for students of universities. Each student can apply and if the school has license with the company, student gets the license which is valid for the time of the student's study².

Every aspect of IntelliJ IDEA is specifically designed to maximize developer productivity. Together, powerful static code analysis and ergonomic design make development not only productive but also an enjoyable experience. (Jet Brains, 2016). One of the strong aspect of IntelliJ is its code completion. During the writing of code, suggestions are showed automatically and are always refreshed based on what is the most convenient suggestion. The good thing about code completion is that it provides only expected types.

Another good thing is TODO management which is done by adding TODO phrase into Java comments. Using this feature, programmer can simply comment lines of code for future reference. IntelliJ manages these TODOs and provides the list of all of them with the line of code where they are placed. Example can be seen on Figure 18.

¹ This process can be convenient when more teams in different countries are working on same project.

² For more information about student licenses visit the page at link: <https://www.jetbrains.com/student/>

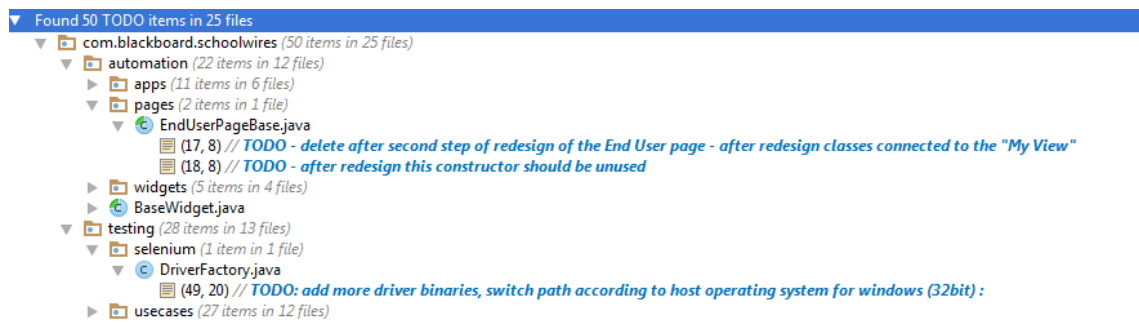


Figure 18 TODO management within the IntelliJ

IntelliJ provides more good features, but it would not be beneficial to list them all. Nevertheless, some of them will be described in the next subsections where another tools will be described and those features will be about the integration of another used tools with IntelliJ.

6.3 Maven

Maven is primarily a build tool for Java projects. Java projects can generate different types of binaries. Typically, the output of a Java project is a JAR file. For web applications, Java classes combined with other type of files result in a WAR or EAR file as well. Maven provides plugins and lifecycle phases to generate various types of binary artifacts for Java projects. The default type of artifact generated by Maven is JAR. If the packaging element is absent, or specified as jar, Maven considers it a JAR project. A JAR project combines all the source classes along with the necessary project resources to a single file. This JAR file can be distributed for it to be used elsewhere. (Bharathan, 2015).

Citation above can sound complicated but the opposite is true. Maven was not made to make work harder. It truly makes the work easier. Very nice feature which Maven provides are project dependencies. Dependencies are all external JARs used within the project. Just for simple illustration, with using maven user don't have to download JAR in order to use it in project. What needs to be done is only add dependency to project `pom.xml` file. This file is generated by Maven and contains project build properties.

For better understanding of dependency injection, on Figure 19 there is example of selenium adding to project. All maven dependencies available can be listed at link: <http://mvnrepository.com/>. When dependency injection is added to `pom.xml` file, project need to be updated and within the project update, maven checks changes in `pom.xml` file and download all added dependencies. This can work also the other way³.

³ When dependency is removed, maven removes JARs that belongs to it

Dependencies can have huge impact on team development. Within the adding of new dependency, all that needs to be done is change pom.xml file, which is very small step and each member of team can do so very fast. But there is no need to do so thanks to version control system. For information of version control system go to 6.6.

```
<!-- minimal required version of Java is 1.8 -->
<properties>
  <sourceJdk>1.8</sourceJdk>
  <targetJdk>1.8</targetJdk>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.53.0</version>
  </dependency>

  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-firefox-driver</artifactId>
    <version>2.53.0</version>
  </dependency>
</dependencies>
```

Figure 19 Maven dependency example

6.4 Page object pattern

Page Object is a Design Pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication. A page object is an object-oriented class that serves as an interface to a page of your automated unit test. The tests then use the methods of this page object class whenever they need to interact with the UI of that page. The benefit is that if the UI changes for the page, the tests themselves don't need to change, only the code within the page object needs to change. Subsequently all changes to support that new UI are located in one place. (Selenium, 2016).

Selenium also comes with solution how to perform page object pattern. It is done by defining functional class and creating web element variables. After it is done, each variable needs to be annotated with `@FindBy` annotation. This tells selenium how to find element on the page. Annotation `@FindBy` takes argument which tells how the element should be found on page. This can be done with element ID, xpath, css or link text. When page object should be used in test, it only takes to create new instance of the class representing page object. Example of google home page is defined by page object pattern can be seen at Figure 20.


```
public class Google {
    @FindBy(id = "lst-ib")
    private WebElement searchInput;

    @FindBy(name = "btnK")
    private WebElement searchButton;

    /**
     * Search given value in google
     *
     * @param value value to search
     */
    public void search(String value) {
        searchInput.sendKeys(value);
        searchButton.click();
    }
}
```

Figure 20 Google home page by page object pattern

Within example project, solution provided by Selenium is not used but APORT framework is used instead. This abbreviation stands for advance page object resolver toolkit. It is a little bit different from Selenium. For starter, it uses different annotations. Annotation `@FindBy` is replaced with `@Resolve`. This annotation takes `value` parameter which tells how to find element on page and `groups` parameter which tells to which group the web element belongs. Page objects are not initialized while instantiating, using this framework. Whole page object initialization handle method `resolve()` which is implemented in `Resolvable` interface. Method takes as parameter list of groups. When `resolve` method is called, it searches for elements belonging to groups given as parameters of the method. Example of reflection of page object and its represented class is possible at Figure 21.



Figure 21 Page object representation using APORT

Another important interface which framework defines is `Page`. `Page` can be instantiated and has to contain specific url. Url figures as unique parameter in this case. It is possible to compose a page using already defined page objects, which can be common thing considering how the web pages works nowadays.

APORT provides good way of resolving page objects but it is possible to customize it specifically in order to meet the wanted goal. In our particular case, there is abstract class `BaseWidget` which represents every single page object. This class containing abstract method `init()`. Each defined page object is extended from `BaseWidget`. This means each defined page object have to override `init()` method. Ideology is that in `init()` method, `resolve()` is called. Using this approach, `resolve()` method is hidden inside the class and only initialization is called in test methods or in places where the page object is needed. Example of page object extended from `BaseWidget` can look like at Figure 22.

```
public class Google extends BaseWidget{

    @ResolveBy(value = "id=lst-ib", groups = "google")
    private WebElement searchInput;

    @ResolveBy(value = "name=btnK", groups = "google")
    private WebElement searchButton;

}

/**
 * Search given value in google
 *
 * @param value value to search
 */
public void search(String value) {
    searchInput.sendKeys(value);
    searchButton.click();
}

@Override
public void init() {
    resolve("google");
}
}
```

Figure 22 Google page described by APORT

By using page object pattern, all page objects can be encapsulated into functional classes. These classes will contain all elements and functions which the page object is providing. It was mentioned before that selenium handles finding of elements. What was not told, that it lets us to do actions with found elements. Web element is an interface provided by selenium and it contains plenty of functions which can be used. The most useful one is function `click`. It simply clicks on an element. The web element interface provides nearly every action that a user can do with elements. Web element is a very general unit and it can represent any element of the DOM. Selenium also provides other interfaces for special HTML elements, like selectors, inputs etc. It is important to know that the role of the selenium web driver in functional testing is to perform the most user-like interaction with the tested application.

All these interactions with web elements that selenium provides should be used within the page object's methods. Considering Figure 21, we have a button which says `Save` and `exit`. This should be represented by a method which could be called the same as the button. This function would take a web element representing the button and click on it. Later in an automated test, only the method would be called and all the selenium functionality would be hidden from test classes. And this mentioned example is the output of using the page object pattern.

6.5 Writing automated test

When test case for automation is provided, it is required to check all the page objects which will be used in the given case. It could happen that already defined page objects are needed. In this scenario, reusability of page object will come as a very good advantage of page object pattern. If object does not exist, it should be created and its functionality should be described properly. If all page objects are implemented, writing the automated test should not be a difficult task. Using provided test case at Figure 17, reflecting test case will look like example at Figure 23.

It is now very clear that all the logic that page objects are providing is encapsulated within the page object itself. The test case itself should represent user interaction with tested application. Only confusing aspects of the code provided at Figure 23 can be assertions. There are a lot of framework which are providing assertions. In this particular case, the one from Google Truth is used. For more documentation about Google Truth reader can visit link: <https://github.com/google/truth>. Purpose of the assertions is to compare two values. One value is value provided as test input and the other one is the actual value. Assertions verify that test in current state contains data that it should contains. If not so, there have to be some problem with tested application.

If all test cases that should be automated are done, there is one thing to left and it is to apply code changes into the main test repository. The process continues to next stage of automation and in most cases, it is handled by version control system.

```

@TestCase("C294")
public void insertImage_imageDisplayed() {
    LOG.info("Validation of image inserting - using friendly values. Image should be inserted");

    // Sign in
    signInPage.openAndSignIn(DEFAULT_ADMIN_LOGIN, DEFAULT_ADMIN_PASSWORD);
    siteManagerPage.checkOpened();
    siteManagerPage.resolve();

    // Navigate to flex editor app
    FlexEditorAppSiteManager app = siteManagerPage.getSectionWorkspaceCurrentPagesWidget()
        .editPage(PAGE_NAME)
        .editApplication(APP_INSTANCE_NAME, siteManagerPage.getFlexEditorAppSiteManager());

    // Upload image
    InsertImageDialog insertImageDialog = app.getCuteEditor().insertImage();
    InsertImageDialog.ImageSourcePane imageSourcePane = (InsertImageDialog.ImageSourcePane) insertImageDialog.getPane();
    imageSourcePane.setFile(relativePathToAbsolute(PATH_TO_TESTFILES + RICHARD_POSE_IMAGE_FILE));

    // Set image attributes
    InsertImageDialog.ImageAttributesPane imageAttributesPane = insertImageDialog.next();
    imageAttributesPane.setAlternativeText(ALT_TEXT_SHORT)
        .setWidth(IMAGE_MAXIMUM_WIDTH);

    // Submit dialog
    insertImageDialog.submit();

    // Save editor changes and sign out
    app.save();
    siteManagerPage.topMenuBar.signOut();

    // Open end user page with flex editor app
    endUserPage.openAndInit();
    FlexEditorAppEndUser flexEditorAppEndUser = endUserPage.getPageWidget().getApp(FlexEditorAppEndUser.class);

    // Validate image is visible
    assertThat(flexEditorAppEndUser.getContentImages()).isNotEmpty();
    assertThat(flexEditorAppEndUser.getContentImages().get(0).getAltText().trim()).isEqualTo(ALT_TEXT_SHORT);
    assertThat(flexEditorAppEndUser.getContentImages().get(0).getWidth()).isEqualTo(IMAGE_MAXIMUM_WIDTH);
}

```

Figure 23 Automated test case from Figure 17

6.6 Version control

Version control is the management of changes in document, computer program or another source with data value. On the market, there are plenty of solutions available. Each one contains different sets of features and each one can be adequate for each situation. In this example, focus will be given on Git. But first, it is important to describe why version control is so helpful and what advantages it brings to the table.

Version control has huge impact on work in team or group of people. It stores information about changes that were made, so it is not a big problem to search who did some specific change, when he did it and what the change was. IntelliJ provides very friendly feature called `annotate`. User of IntelliJ can annotate all lines of code and then see all information described above.

Changes within the version control system have to be grouped somehow into some entity. This entity is usually called commit. Commit contains message, description and changes itself. All the commits are ordered by date. There is always possibility to checkout some of the listed commits. If do so, all the code changes from the commit which was checkout will be reverted. This can be very useful if someone commit some dangerous code.

To prevent committing dangerous code or pieces of code, there is code review. Purpose of code review is to review the changes that were made. If the member of team is not satisfied with code changes or have some advices, complaints etc. he can raise a request or comment to change it. On the other hand, if he is satisfied he can approve it giving permission to project maintainer to merge the changes into the upstream. In this scenario, changes were successfully made and changes were successfully merged into the upstream. Then it is responsibility of each team member to update latest version of the code.

As it was mentioned before, our focus will be to bring closer look at Git workflow and how the Git works with IntelliJ integration. Scheme of Git workflow is illustrated on Figure 24. In the earlier provided example, automation engineer finished his automated tests and wanted to apply his changes to the code. Using IntelliJ, all that is required to do is raise a commit request. After that, all changes are committed into the engineer's local repository. To push it to remote repository, there is push request. From this point, all that is left to do is to create code review and when it is done, all the changes can be merged. New automated tests was added to the project and process can repeat again.

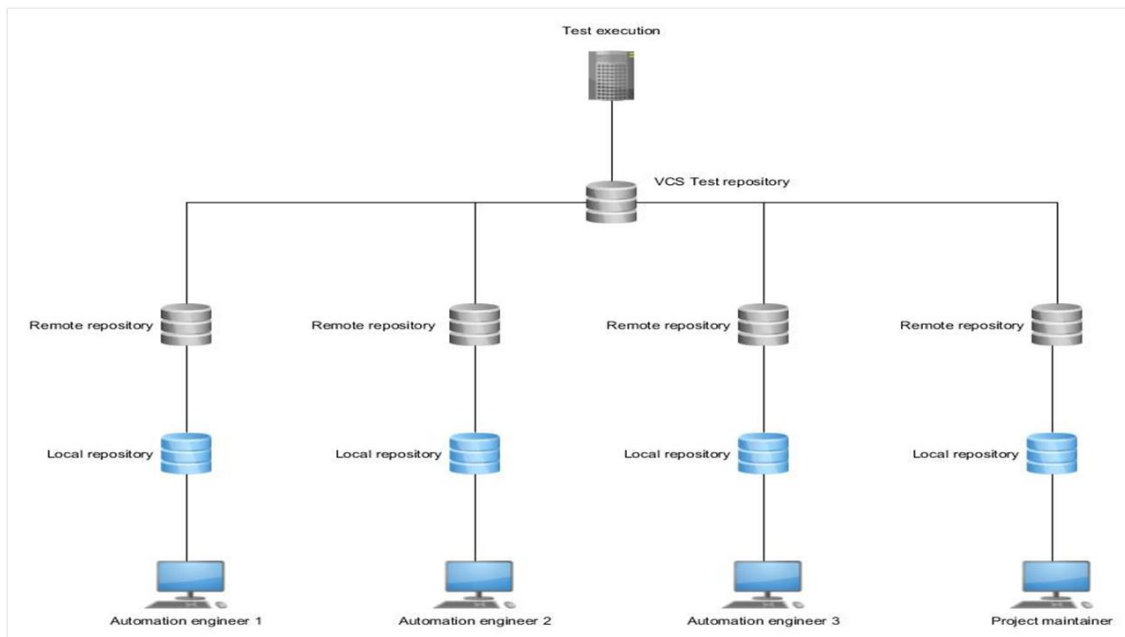


Figure 24 Git workflow in test automation

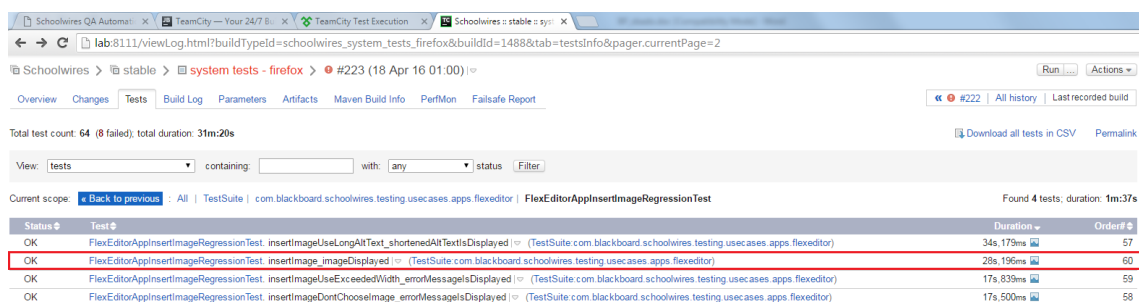
6.7 Test execution and reporting

If the whole process should be automated, it would be appropriate if the test execution can be done independently and without interference from outside sources. This can be done by build agents. As it says in the name, these agents are able to build code files. As it is illustrated at Figure 24, there is test repository which con-

tains build of test project. Above that is some box called test execution. And that is exactly build agent which takes last build from test repository and execute it.

It provides much more than just building application. It is possible to set when the build should be done, it is possible to provide parameters of the build, see changes that were made since last build, create report, etc. It also provides build log information which in case of tests can be test output from logging. In this case, as build agent figures TeamCity from Jet Brains. For more information about TeamCity itself, visit the link: <https://www.jetbrains.com/teamcity/>.

When test run is complete, information about build is possible within TeamCity. Now would be a good time to check if the previously created test passed and if everything is okay. At Figure 25, we can see that our highlighted test was completed successfully.



Status	Test	Duration	Order
OK	FlexEditorAppInsertImageRegressionTest.insertImageUseLongAltText_shortenedAltTextsDisplayed (TestSuite:com.blackboard.schoolwires.testing.usecases.apps.flexeditor)	34s.179ms	57
OK	FlexEditorAppInsertImageRegressionTest.insertImage_imageDisplayed (TestSuite:com.blackboard.schoolwires.testing.usecases.apps.flexeditor)	28s.195ms	60
OK	FlexEditorAppInsertImageRegressionTest.insertImageUseExceededWidth_errorMessagesDisplayed (TestSuite:com.blackboard.schoolwires.testing.usecases.apps.flexeditor)	17s.839ms	59
OK	FlexEditorAppInsertImageRegressionTest.insertImageDontChooseImage_errorMessagesDisplayed (TestSuite:com.blackboard.schoolwires.testing.usecases.apps.flexeditor)	17s.500ms	58

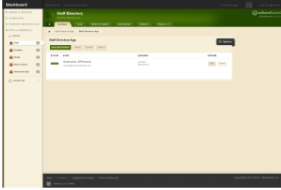
Figure 25 Test report provided by TeamCity

Failures in tests can be caused by bugs in tested applications or by errors in automated test. It can happen that test which worked before can fail in slower environment. Failures can be also caused by timeouts in tested application. However, these failures and causes of failures can be provided also. It is done by reporting service. It is possibility of choice which service to use. It was mentioned before that TestNG is used as test unit framework and it contains plugin called ReportNG. See: <http://reportng.uncommons.org/>. Work of ReportNG is to provide test output, so it exists some feedback. In test output, causes of failures can be found and, failures can be analyzed and possible fixes can be introduced.

Report example generated by ReportNG is illustrated at Figure 25. Test cases are grouped into functional wholes and this whole is represented by JAVA class in this scenario. At the figure, it is possible to see what information ReportNG provides. Now is the time to determine what the cause of failure was. If it was not bug, some fixes in the code needs to be made. On the other hand, if the test found bug, information about the bug have to be provided to developers so possible bug fixes can be made. With the exception of analyzing test failures and refactoring of the code, test development process will start again from the beginning and all its phases will be repeated again.

com.blackboard.schoolwires.testing.usecases.apps.staffdirectory.StaffDirectoryAppRegressionTest
staffDirectoryRegressionTest 24.149s test cases C170

staffDirectoryRegressionTest



org.openqa.selenium.StaleElementReferenceException: Element not found in the cache – perhaps the page has changed since it was looked up
For documentation on this error, please visit: http://seleniumhq.org/exceptions/stale_element_reference.html
Build info: version: '2.53.0', revision: '35ae25b', time: '2016-03-15 17:00:58'
System info: host: 'bfu-VirtualBox', ip: '127.0.1.1', os.name: 'Linux', os.arch: 'amd64', os.version: '4.2.0-18-generic', java.version: '1.7.0_80'
Driver info: driver.version: unknown
Command duration or timeout: 1.52 seconds
For documentation on this error, please visit: http://seleniumhq.org/exceptions/stale_element_reference.html
Build info: version: '2.53.0', revision: '35ae25b1534ae328c771e0856c93e187490ca824', time: '2016-03-15 10:43:46'
System info: host: 'tab2', ip: '122.0.1.1', os.name: 'Linux', os.arch: 'amd64', os.version: '3.19.0-15-generic', java.version: '1.8.0_72'
Driver info: org.openqa.selenium.remote.RemoteWebDriver
Capabilities {applicationCacheEnabled=true, rotatable=false, databaseEnabled=true, handlesAlerts=true, version=45.0.1, platform=LINUX, nativeEvents=false, acceptSslCerts=true, webdriver.remote.sessionid=3b37c9b5-19aa-47de-80d0-0536e2f96a95, locationContextEnabled=true, webStorageEnabled=true, browserName=firefox, takesScreenshot=true, javascriptEnabled=true, cssSelectorsEnabled=true}
Session ID: 3b37c9b5-19aa-47de-80d0-0536e2f96a95

Figure 26 Example of reported test failure

7 Discussion

Test suite containing automated test cases covers certain part of whole tested software functionality but it can be and also will be continuously expanded with more automated tests. In this chapter, solution proposed in the thesis will be evaluated from economic point of view. Whole economic evaluation is based on document written by Douglas Hoffman. (Hoffman, 2016).

7.1 Economic evaluation

When considering test automation, it is important to determine whether automation will lead into savings of money. But before we can proceed to calculations, it is important to describe the costs related to test automation. Hoffman distinguishes between fixed and variable automation costs. Examples of fixed automation costs can be hardware, software licenses, scripting tools, tools training, etc. Variable costs are test case implementation, test maintenance, test execution, analysis of test results, reporting, etc.

To calculate how the test automation will contribute, following equation can be used.

$$(1)ROI_{automation} = \frac{B_a}{C_a}$$

$$(2)ROI_{automation} = \frac{\Delta B_a}{\Delta C_a}$$

ROI = return of investment

B_a = the benefits from automated testing

C_a = the costs of automated testing

δB_a = benefits from automated over manual testing

δC_a = costs of automated over manual testing

Equation (1) shows how the ROI can be computed in a general form. It is the ratio of the benefits from automation over their costs. A value greater than one indicates that the benefits are greater than the costs. It shows that for every dollar invested in automation we get ROI dollars, worth of benefits. (Hoffman, 2016).

Equation (2) shows a relative ROI for comparing the added benefits from automation with the added costs from automating. Although the equation is very general, it more reasonably represents the value of automation in relation to manual testing. (Hoffman, 2016).

For purpose of this evaluation, equation (2) will be used. Benefits from automated testing over manual testing are calculated as improvement in fixed costs of automated testing times + variable costs of running manual tests during time - variable costs of running automated tests during time. Costs of automated over manual testing are calculated as increased fixed costs of automated testing times + variable costs of creating automated tests - variable costs of creating manual tests + variable costs of maintaining automated tests during time.

Using this knowledge, we can calculate benefits from automation using following assumptions:

- A product with new tests
- 5 people developing manual tests
- 10 people running manual tests
- 5 people developing automated tests(include also maintenance)
- Automated tests run automatically
- 1 people environment maintenance
- Periods of time selected: 24 months
- People cost 320 000 CZK per year = 170 CZK per hour
- Fixed costs for automated tests 110 000 CZK

$$\Delta B_a(24) = 0 + (10 * 640,000) - (0 * 640,000) = 6,400,000$$

$$\Delta C_a(24) = (110,000 * (2/3)) + (5 * 640,000) - (5 * 640,000) + (5 * 640,000) = 3,273,000$$

From this example, we can assume that ROI factor will have value of 1.95. This means that 1 CZK invested to automation will return 1.95 CZK which is nearly 100% economic return. In this particular example, test automation will be effective. However, in some cases automation is not appropriate and can lead into more financial costs.

8 Conclusion

In introduction, software testing was introduced, different testing levels and testing approaches were described. There was a lot of theoretical information which are needed and without it, it would be hard to understand following chapters of this thesis.

Next chapter and its focus was given to tools used for test automation. These tools were differenced by theirs purpose. It was obvious that there are plenty of tools available and it can be very hard task to choose one of them. Sometimes it can be bad choice and customers can regret their choice. To prevent this situation, best solution is to analyze the tools properly. A lot of commercial tools were provided as trials so it should not be a problem to test first and choose after.

As next was described tested application. It is always better to understand what will be tested so reader can have better understating of what is going on. Only key concepts of the tested applications were described and this was on purpose, because these functionalities were only tested as well.

As the next step, whole process of writing automated tests was described. This process is also applied at Blackboard Czech s.r.o which is owner of the test project. Whole test project is contained at additional CD with its execution report. Execution was done using two different browsers (Firefox and Chrome). It was discovered that browser has only time affection to test execution and it was obvious that Chrome is much faster than Firefox.

Personal benefits are better understanding of how is software testing so important and perspective about which tools are available to perform automation. Another benefit was better understanding of object-oriented programming and how to use it. Analytical skills were improved as well.

With described test execution management, test suite can be execute properly and it is controlled by TeamCity. Execution is scheduled on each day and test results are always analyzed. Biggest advantage of scheduled test execution is that it is performed every day and it is independent on other factors.

Future plan of quality assurance team is to provide more and more automated tests. Number of apps within the Centricity2 is big and all of them needs to be covered. If all the apps will be covered, possible apps interactions with each other can be tested as well.

9 Literature

- AMMANN, P. JEFF, O. *Introduction to software testing*. New York: Cambridge University Press. 2008. 322 p. ISBN 0521880386.
- BHARATHAN, R. *Apache Maven Cookbook*. Birmingham: Packt Publishing Ltd. 2015. 272 p. ISBN 978-1-78528-612-4.
- BRYNDZA, M. *Automatizácia testov korektnosti a regresných testov tlačového riešenia YSoft SafeQ*. Brno. Bachelor thesis. 2012
- COHEN, F. *Java testing and design: from unit testing to automated Web tests*. 2nd edition. Upper Saddle River, NJ: Prentice Hall PTR. 2004. ISBN 01-314-2189-1.
- COPELAND, L. *A practitioner's guide to software test design*. London: Artech House. 2004.
- GAJDOŠÍK, M. *Automatizované testy bankovního softwaru*. Brno. Bachelor thesis. 2015
- GUROCK, *TestRail* [online]. [cit. 2016-04-14]. Available at: <http://www.gurock.com/testrail/>
- HOFFMAN, D. *Cost Benefits Analysis of Test Automation*. [online]. [cit. 2016-05-06]. Available at: <http://www.softwarequalitymethods.com/Slides/AutoCost%20Slides.pdf>
- CHELIMSKY, D. *Random thoughtlessness* [online]. [cit. 2016-03-21]. Available at: <http://blog.davidchelimsky.net/blog/2007/05/14/an-introduction-to-rspec-part-i/>
- JET BRAINS: *IntelliJ IDEA* [online]. [cit. 2016-04-16]. Available at: <https://www.jetbrains.com/idea/>
- JET BRAINS: *TeamCity* [online]. [cit. 2016-04-16]. Available at: <https://www.jetbrains.com/teamcity/>
- KANGLIN, L. MENGQI, W. *Effective GUI test automation: developing an automated GUI testing tool*. San Francisco, Calif: SYBEX. 2014. ISBN 07-821-4351-2.
- KOLAWA, A. *Regression testing*. [online]. [cit. 2016-02-27]. Available at: <http://www.wrox.com/WileyCDA/Section/id-291252.html>
- LABOON, B. *A friendly introduction to software testing*. 1st edition. CreateSpace Independent Publishing Platform. 2016. ISBN 978-1523477371.
- PATTON, R. *Software testing*. 2nd edition. Indianapolis, IN: Sams Pub. 2002. ISBN 06-723-2798-8.
- RAILSHORDE *Waterfall Development Model* [online]. [cit. 2016-02-28]. Available at: <http://www.railshorde.com/blog/waterfall-development-model>
- RANOREX [online]. [cit. 2016-03-29]. Available at: <http://www.ranorex.com/>
- RÁBOVÁ, I. *Podnikové informační systémy a technologie jejich vývoje*. Brno. Tribun EU. 2008. ISBN 978-80-7399-599-7.
- RSPEC [online]. [cit. 2016-03-13]. Available at: <http://rspec.info/>

SELENIUM, *Selenium documentation*. [online]. [cit. 2016-04-16]. Available at: <http://docs.seleniumhq.org/docs/>

SHORE, J. WARDEN, S. *The art of agile development*. Sebastopol. CA: O'Reilly Media, Inc., 2008. ISBN 9780596527679.

SMARTBEAR, *SoapUI* [online]. [cit. 2016-03-22]. Available at: <https://www.soapui.org/>

TESTNG [online]. [cit. 2016-03-13]. Available at: <http://testng.org/>

Appendices

A Additional CD

Content of the additional CD:

Whole project containing automated tests - `schoolwires-systemtests`

Reports from test runs on Chrome and Mozilla Firefox - `test-report`

Recorded run of example automated test `example-test-run`

