




TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

DMITRII ROGOZIN A TESTING TOOL FOR VAADIN APPLICATIONS

Master of Science thesis

Examiner: Prof. Kari SystÄ 
Examiner and topic approved by the
Faculty Council of the Faculty of
xxxx
on 1st November 2015

ABSTRACT

DMITRII ROGOZIN: A testing tool for Vaadin applications

Tampere University of Technology

Master of Science thesis, xx pages, x Appendix pages

xxxxxx 201x

Master's Degree Programme in xxx Technology

Major:

Examiner: Prof. Kari SystÄ

Keywords:

The abstract is a concise 1-page description of the work: what was the problem, what was done, and what are the results. Do not include charts or tables in the abstract.(100-150 words)

PREFACE

This document template conforms to Guide to Writing a Thesis at Tampere University of Technology (2014) and is based on the previous template. The main purpose is to show how the theses are formatted using LaTeX (or L^AT_EX to be extra fancy) .

The thesis text is written into file `d_tyo.tex`, whereas `tutthesis.cls` contains the formatting instructions. Both files include lots of comments (start with `%`) that should help in using LaTeX. TUT specific formatting is done by additional settings on top of the original `report.cls` class file. This example needs few additional files: TUT logo, example figure, example code, as well as example bibliography and its formatting (`.bst`) An example makefile is provided for those preferring command line. You are encouraged to comment your work and to keep the length of lines moderate, e.g. <80 characters. In Emacs, you can use `Alt-Q` to break long lines in a paragraph and `Tab` to indent commands (e.g. inside figure and table environments). Moreover, tex files are well suited for versioning systems, such as Subversion or Git.

Acknowledgements to those who contributed to the thesis are generally presented in the preface. It is not appropriate to criticize anyone in the preface, even though the preface will not affect your grade. The preface must fit on one page. Add the date, after which you have not made any revisions to the text, at the end of the preface.

Tampere, 1.9.2015

Dmitrii Rogozin

TABLE OF CONTENTS

1. Introduction	1
2. Theoretical background	3
2.1 Web applications	3
2.2 Vaadin	4
2.3 Testing	6
3. Web testing	8
3.1 Capture and Replay	8
3.2 Programmable tests	9
3.3 Capture and Replay vs Programmable tests	9
3.4 Challenges	12
3.4.1 Look and feel testing	12
3.4.2 Complex DOM structure	12
4. Selenium	15
5. Reason for developing Testbench	19
5.1 Client server communication	19
5.2 Extra code	19
6. Development of Testbench	20
6.1 Scrum	20
6.2 Test Driven Development	22
6.3 Tools used	22
6.3.1 Maven	22
6.3.2 Trac	24
6.3.3 Git	25
6.3.4 Teamcity	26
6.3.5 Gerrit	27

6.4 Testbench class diagram	28
6.5 Basic test case structure	31
6.6 Results	32
7. Testbench vs Selenium	33
7.1 API built specifically for Vaadin components	33
7.2 Client server communication	33
7.3 Screenshot comparison	34
7.4 Parallel testing	36
8. Testbench use	38
8.1 Integrating with Behaviour-Driven Development frameworks	40
9. Conclusion	42
9.1 Discussion and Conclusion	43
9.1.1 Summary	43
9.1.2 Advantages and disadvantages	43
9.1.3 Future work	43
.1 Appendix A	46
.1 Appendix B	52

LIST OF FIGURES

2.1 Web application structure	4
3.1 Gmail page structure	13
4.1 Web structure	16
4.2 Selenium Grid structure	17
6.1 Gerrit structure	28
6.2 Testbench class diagram	29
7.1 Client server synchronization in TestBench	34
7.2 Reference screenshot	35
7.3 Screenshot with emphasized error	35
8.1 Table component extension example.	39

LIST OF TABLES

1	List of closed tickets.	53
---	-------------------------	----

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
BDD	Behaviour-Driven Development
CVAL	Commercial Vaadin Add-On License
DOM	Document Object Model
GWT	Google Web Toolkit
JVM	Java Virtual Machine
HTML	HyperText Markup Language
XHTML	EXtensible HyperText Markup Language
C&R	Capture and Replay
IDE	Integrated Development Environment
JRE	Java Runtime Environment
POM	Project Object Model
UI	User Interface
URL	Uniform Resource Locator

1. INTRODUCTION

The 21st century has become an era of Web applications. Software systems developed as a Web based application allows the end user to access data via Web browser from different parts of the world and also from different devices (laptops, phones, tablets). Ability to access an application from different places and devices, without need to install any additional software, has become the main feature of modern applications. Web applications reduces a complexity of accessing products and services and make them more attractive to an end user.

Static HTML Web sites, with a little amount of Javascript, which were constituting the big part of the Web are losing their popularity. Modern Web applications are very interactive and dynamic, they are becoming more powerful, and the difference between desktop and Web applications disappears.

Web technologies are developing so fast, that even such domain specific applications as Integrated Development Environments (IDE), trading systems or graphic editors can be accessed via Web browser. As a result of the growth of Web applications, development and maintenance of such complicated systems becomes more challenging. To reduce development and maintenance costs software frameworks are coming into existence.

In the thesis I will describe tools and methodologies used for Web testing. Mainly I will focus on a Java-base framework for developing Web applications called Vaadin and a testing tool called Vaadin Testbench.

In the Fall of 2014 I was a part of the team which developed Vaadin Testbench 4.0.0 and released it in the beginning of December. Web testing tools is a new topic and I will represent the main ideas and challenges of Web testing and how they were solved during Testbench development.



The goal of this work is to provide a tool for a developer, that will help to write

tests which can simulate user actions on the Web page. The main challenge is that code written with Vaadin executes both on the client-side and server-side and testing tool should handle all the complexity in testing client-side and server-side communications. Another challenge is to develop an universal and easy to use testing tool for Vaadin framework with a clear API.

The result of the work was Testbench 4.0.0 released in December 2014. Several user tests have shown, that a person with experience in Java and Vaadin, but without any experience using Testbench, needs 15 minutes to setup the environment and write a simple “button-click” test. We consider this result as a success.

Vaadin is an open-source project, but Testbench is a commercial addon, nevertheless, there is a 30 days trial period, so anyone can try Testbench and take a look on results of our work.

The thesis is structured as following. Chapter 3 describes and compares different techniques used in Web testing. Chapter 2.2 describes Vaadin framework. Chapter 6 presents the developing process of TestBench tools and methodologies used. Chapter 8 shows the example of writing tests and the value for end user from using Testbench. Chapter 9 summarize the whole thesis and list the results.



2. THEORETICAL BACKGROUND

2.1 Web applications

Static HTML Web sites are losing their popularity, because users expect from modern Web sites more than just representing pictures and text. Generally, users want to have a highly responsive application with different useful features, working in the Web. As a result Web applications are displacing Web sites on the market.

The difference of Web application from a Web site is the “ability of a user to affect the state of the business logic on the server”[7]. In other words a user or client makes a request to the server, the server performs some actions (calculate, fetch data from database or external Web-service) and sends the response back to the client, which is rendered in the browser see figure 2.1

Client-server structure helps to distribute application tasks or workloads between the service providers called servers, and service requestors, called clients. Client-server model helps to separate client and server logic, as a result these parts can be independent and communicate via Application Programming Interface (API). Client server model grants several advantages:

- Client and server parts can be developed separately.
- The application may have several clients.
- A client, for example Web browser, can be already created.

Client server model is closely related to a multi-tier architecture - the concept where the parts of the system are divided into separate tiers. Web applications are often use three-tier architecture:

- Presentation tier is responsible for user interface generation and lightweight validation.

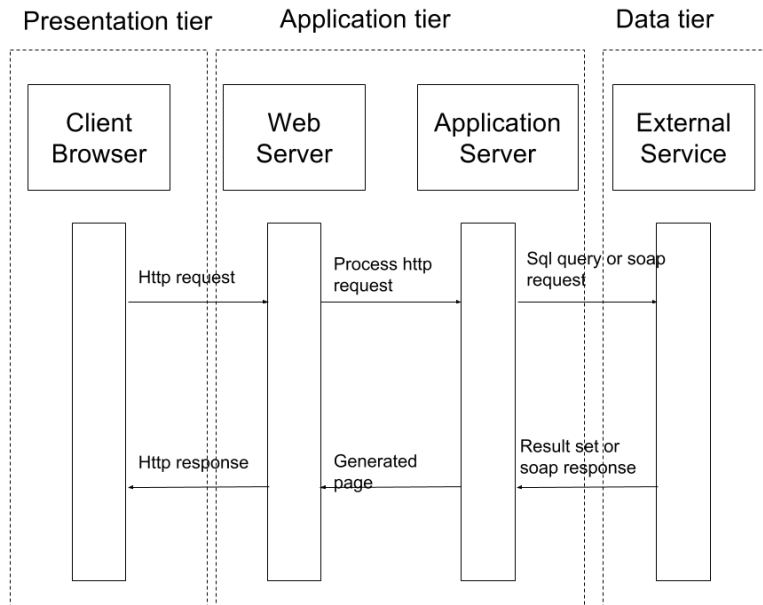


Figure 2.1 Web application structure

- Application tier (business logic tier) controls an application functionality, determines how data is created, displayed, stored and changed.
- Data tier - controls databases or other resources, provides access to the data.

Multi-tier architecture allows any of the three tiers to be changed or replaced independently, as a result developers have more freedom to use external libraries and frameworks.

All applications have a lot of common features and problems which were already solved by developers beforehand. It is a good practise to take an already existing solution, than try to implement your own new one. That is why many modern applications are based on one or several software frameworks. In a rapidly changing and highly competitive business environment, choosing a right toolset is one of the key factors of the success.

2.2 Vaadin



“Vaadin Framework is a Java Web application development framework that is designed to make creation and maintenance of high quality Web-based user interfaces easy. Vaadin supports two different programming models: server-side and

client-side. “ [1] pr1.1] *Client-side* Vaadin code is executed in the Web browser as JavaScript code. *Server-side* code is executed on the server as Java code on the Java Virtual machine(JVM).

Client-side code is originally written in Java and then compiled to JavaScript using *Vaadin Client Compiler*. Vaadin Client Compiler is based on Google Web Toolkt(GWT). The client-side code is responsible for rendering the user interface and send user interaction to the server. Vaadin Client Compiler has production and development modes.

In production mode Java client-side code is compiled to one Javascript file. The script file in production mode is obfuscated, that is why reading or debugging it is nearly impossible. In production mode compilation should be called manually, after making changes to a client side Java code. In development mode client code is compiled at run-time, when Web page is reloaded. GWT links Java classes with a compiled Javascript and gives an opportunity to debug code in a browser.

Nowadays there are a lot of standards and recommendations for Web developers published by World Wide Web Consortium (W3C) or International Organization for Standardization (ISO), including recommendations for markup languages (HTML, XML), Document Object Models(DOM) and standards for JavaScript. In spite of all the standards the difference between browsers and versions might be significant for the developer. The differences may vary from supporting/not supporting different Cascading Style Sheets (CSS) tags and HTML5 features, different event handling and simply bugs. Vaadin client Compiler and GWT provide a wide browser support, eliminating the difference between browsers, and helping a developer to concentrate on essential parts of the application, instead of wasting time on cross-browser support. Vaadin uses screenshot comparison see [7.3] as a part of regression testing which brings confidence to the developer that Vaadin components will not change their appearance unexpectedly.

A server-side code runs as a servlet in a Java app server, serving HTTP requests. The VaadinServlet is normally used as the servlet class. The servlet receives client requests and interprets them as events for a particular user session. Events are associated with user interface components and delivered to the event listeners defined in the application. If the User Interface (UI) logic makes changes to the server-side user interface components, the servlet renders them in the Web browser by generating a response.





As mentioned before both client-side and server-side code in Vaadin is written in Java. This positively influences the development process in the following way:

- The developer does not need to know several programming languages and one person may be involved in the developing of both front-end and back-end. This might be an important factor for small teams and speeds up the development process.
- Vaadin brings the power of Java into the Web development. Due to TIOBE index [2] Java and C are the two most popular programming languages since 2002. This fact allows developer to use a great amount of already-made solutions such as building tools Maven and Ant, testing tools, frameworks for concurrent applications as Akka and other libraries like Yodatetime, Guava for different stages of development process [3, 4, 5, 6, 7, 8].

2.3 Testing

Nowadays some companies still rely on manual testing or ignore this important part of software development ~~at all~~. Such approach has several sorrowful consequences:



- The developers are afraid of changing already written code. Because they do not have a confidence that their changes would not break existing code. They stop cleaning their production code because they fear the changes would do more harm than good. "Their production code begin to rot" [9, p.123]
- The effort of finding errors and fixing them raises with the amount of code written, because the developers can not localize the place where the error is actual happening.
- Developing new features becomes harder, if they are based on the part of the system which have errors.
- Costs the whole system increases.

To test easily the huge amount of code an automated testing is needed. Automated testing reduces the amount of work required to check Web applications as well as Web sites, amplify software value, enhance reusability of test cases and improve time-to-market.

IEEE defines software testing as the process of evaluating a software system to verify that it satisfies specified requirements [10]. A set of requirements for a Web application includes security, performance, presentation, etc. We will focus on several requirements for the Web application which differ from desktop application.

One of the key requirements which makes testing Web applications harder than testing desktop application is support of different browsers and operating systems and also different devices. A lot of desktop applications are developed to support some particular operating system or different versions of the product are developed and maintained for different operating systems.

Web applications on the contrary should support not only different operating systems, but also different browsers and devices. So, if developers team decides to support three operating systems (Windows, OSX, Android), three type of devices (phone, tablet, PC) and three browsers (Chrome, Firefox, Internet Explorer) the number of possible variations is already twenty seven. If you decide to support different version of browsers, which in some circumstances may vary a lot, the number of different configurations of tested machines will be close to one hundred. In this case manual testing is unexceptable, because it will lead to unwarranted expenses.

Another difference between Web and desktop applications is navigation on the Web page and between pages, the unexpected state change via the browser back button or direct Uniform Resource Locator(URL) entry in the browser. Some resources or parts of the application can be not accessible, due to connection problems or maintenance. Such unexpected behaviour may happen, and must be handled properly, not to crash the whole application.

As mentioned previously one of the key factors for a successful development process is to pick a right toolset, this also applies to testing. In the chapter 3 we will present work done by other researchers and show how we can use it in creating a test framework for Vaadin applications.

3. WEB TESTING

There are several approaches for Web testing, the choice among them depends on different factors such as lifecycle of the project, technologies used, the budget, the professional level of developers. Two main ideas are Capture and Replay tests (C&R) and programmable tests.

3.1 Capture and Replay

C&R Web testing is based on capture and replay automated tools. The software tester works with the Web application modeling user behaviour, the capture/replay tool records the whole session and generates the script, which can be executed later, repeating same actions without human participation. The main idea of C&R testing tools is to record a sequence of browsing steps, that can be automatically replayed later and save them in human readable format. Human readable format is an important feature, because script editing might be useful to adjust failed scripts accordingly to the changes of the Web page. Thought if the Web page was changed significantly, editing test script might be more expensive than recording a new test[11].

C&R tools are usually Web browser plugins or Java applets, because they need to control a Web browser to navigate on a Web page. User actions are usually saved as a sequence of steps in HTML or XML, which is later used to generate a Javascript to reproduce user actions. An example of C&R testing tool is Selenium IDE described in chapter4.

The main advantage of C&R, is that the tester does not require to have experience in coding. Building test cases with such tools is a simple task. On the contrary, maintaining C&R tests is harder and more expensive. The main problem is that editing generated scripts is harder than editing scripts written by a software developer. The test cases are strongly coupled with Web pages and contain hard-coded

values. These factors lead to the problem, that very often the tester have to record a new test, instead of changing the existing ones.

When using C&R tools the tester can not use loose-coupling and decomposition, and other design techniques to make easy readable and maintainable tests. It is also hard to use parts of already made test cases when creating new ones. Programmable tests can help to solve these problems.

3.2 Programmable tests

Programmable tests are created by a tester manually. This method requires the person to have programming skills and takes more time, but programmable tests are more flexible and allow the developer to use bigger set of tools. The developer may use conditional statements to change execution of the test, loops to repeat same actions, exception handling, data structures like arrays, sets, trees, graphs, logging and etc. Programmable tests are more flexible and powerful than C&R tests and provide an opportunity to create parameterized tests - tests which can be executed multiple times with different arguments.



Thought writing programmable tests is harder in compare with C&R and requires more experience and skills, programmable tests provide more flexibility and scalability. The empirical study of developing tests for four different frameworks shows that the development of programmable tests is more time consuming (between 32% and 112%), but test maintenance requires less time (with a saving of 16% and 51%). As a result “In general, programmable test cases are more expensive to write but easier to evolve than C&R ones, with an advantage after two releases (in the median case)”.[\[11\]](#)

3.3 Capture and Replay vs Programmable tests

To show that programmable tests are more powerful than C&R we provide an artificial example of testing Vaadin framework. Vaadin has a set of UI element classes and we want to test that changing elements value triggers a value change listener event. First we create a Web page where we add elements we want to test such as text field, combo box, radio button and an assertion input element see TestWebPageClass [3.1](#). UI elements are added to a hash map as keys. Strings which will

be set to the assertion element are added as values to the same map line 5. An assertion element we include a string line 9, which will be compared with expected value. Then we iterate through all values in the map, set their ids line 13 and add elements to the Web page line. We also add value change listeners line 15, which will set the value of the assertion element according to the event triggered. Finally we get a Web page with set of elements. Setting value of an element will set the assertion element to have the id of this element as value. For example if an element with an id “textfield” has changed its value, the assertion element will have string “textfield” as value.

```

1 public class TestWebPageClass {
2     static final String ASSERT_ELEM_ID="assertElementId";
3     static Map<AbstractElement,String> map = new HashMap();
4     static
5         map.put(new TextField(),"textField");
6         map.put(new ComboBox(), "combobox");
7 }
8
9     TextField assertionElement=new TextField();
10 public void createTestWebPage () {
11     Iterator it = classToAssertValue.entrySet().iterator();
12     while (it.hasNext()) {
13         it.getKey().setId(it.getValue());
14         addElementToWebPage(it.getKey());
15         it.getKey().setChangeListener(event-> {
16             assertionElement.setValue(it.getValue());
17         });
18     }
19
20     addAssertElement();
21 }
22
23     public static <AbstractElement,String> getMap() {
24         return map;
25     }
26 }
```

Program 3.1 Test Web Page class

Afterwards we create a test we iterate through all elements in the map and find the element on the test Web page by id 3.2. Then set a value to this element, a value change listener of the element should be triggered and set the value of the assertion element. In the last step we compare the value in the assertion element with a value

in the map.

```

1 public class ValueChangeListenerTestClass {
2     <AbstractElement,String> map=TestWebPageClass.getMap();
3     String assertElementId=TestWebPageClass.ASSERT_ELEM_ID;
4     UIElement assertElement=findElementById(assertElementId);
5
6     @Test
7     public void testValueChangeListener() {
8         openWebPage();
9         Iterator it = map.entrySet().iterator();
10
11         while (it.hasNext()) {
12             Map.Entry pair = (Map.Entry)it.next();
13             UIElement elem=findElementById(map.getValue());
14             elem.setValue('foo');
15             String assertMessage='Element with id='+pair.getValue()
16                 + 'has wrong value';
17
18             Assert.assertEquals(assertMessage,assertElement.getValue(),
19                 pair.getValue());
20         }
21     }
22 }

```

Program 3.2 Test class

As mentioned before, the biggest advantage of programmable tests against C&R is scalability. When using C&R a tester should record same actions for each new element, while with programmable tests, testing new elements requires just adding these elements to the map.

If later we decide to have a test that checks that “getValue()” method returns the same value as it was set with “setValue()” method we can create a new test method which will use the same map of elements see [3.3](#).

```

1 //test getValue() and setValue()
2 @Test
3 public void testSetValue() {
4     private String testValue="foo";
5     openWebPage();
6     Iterator it = map.entrySet().iterator();
7     while (it.hasNext()) {
8         Map.Entry pair = (Map.Entry)it.next();

```

```

9         UIElement elem=findElementById(map.getValue());
10        elem.setValue('foo');
11        Assert.assertEquals(elem.getValue(),testValue);
12    }
13 }


```

Program 3.3 Test class

3.4 Challenges

3.4.1 Look and feel testing



Both C&R and programmable tests are  used on a DOM of the Web page. One disadvantage is they do not provide tools appearance of the Web page like colors, margins, fonts, etc. The client side page may have bugs in CSS or HTML, for example if all HTML elements had CSS rule display:none, they would not be shown for a user in the Web browser. Thought all user actions could be still emulated by a testing tool. This problem can be solved by adding screenshot comparison see [7.3](#), when a screenshot of a tested Web page is compared with a reference screenshot.

3.4.2 Complex DOM structure

Real-life example may have dozens/hundreds of HTML elements on the Web page see figure [3.1](#). Web pages with big and branched DOM bring several challenges:

1. Searching for required element may be very resource-consuming, affecting time of the test execution.
2. Changes in DOM may require changes in tests, which increase application maintenance costs.

Testing frameworks allow several strategies for locating Web page elements:

- By id -locates the Web page elements using their id values.
- By name -locates the Web page elements using their name.



Figure 3.1 Gmail page structure

- By tag - locates the Web page elements using their tag.
- By class - locates the Web page elements using their class attribute.
- By XPath - combines previous strategies and builds a search path to an element in the DOM.

The efficiency between these strategies is a **trade off** between efficiency of the test and its complexity for developer. The research of Maurizio Leotta and Diego Clerisipaper in “An Industrial Case Study about Web Page Element Localization” shows that ID-based methods for locating Web page elements are better than XPath methods [12] showing that tests with search by Xpath executed more than three times longer than same tests with searching elements by Id. According to the same paper ID-based test require less maintenance effort, than the XPath-based test suites. In fact using searching by id for static HTML Web pages with small amount of elements works well, but having dynamically generated HTML with a lot of elements as in example 3.1 brings challenges.


The biggest downside of searching by id strategy, is that every HTML element should have a unique ID. If the Web pages has a dynamically generated content, for example a table, where amount of rows depends on data, the developer has to add some logic to generate ID for each row and also verify that new ids do not conflict with already created ones.

In some circumstances the developer needs to get a set of elements by some criteria, for example get all elements with a specific tag or class selector and process them

in a loop.

As we can see there is no one solution for searching elements on a Web page, which can be used in all cases. The developer should make a solution which searching algorithm to use based on requirements, but the testing framework should provide the developer different tools to choose from. In chapter [4](#) we will present Selenium - a software testing framework for Web applications, which allows to create C&R [3.1](#) and programmable tests [3.2](#). Selenium supports searching elements by id,tag, class, XPath, etc and provides an opportunity simulate user actions on a Web page.

4. SELENIUM

Selenium is a set of different software tools each with  a different approach to supporting test automation. The entire suite of tools allows many options for locating UI elements and comparing expected test results against actual application behavior [13]. Selenium provides implementation both of C&R [3.1][3.2] and programmable tests models.

Selenium IDE - is a development environment with graphical interface for building test scripts. Selenium IDE has a recording feature, which records user actions as they are performed and then exports them as a reusable script in one of many programming languages that can be later executed. An example of recorded test [1] opens a URL type two numbers to fields with “number1” and “number2”, then - clicks and “add” button and verify the result see Appendix [1].

Selenium WebDriver makes direct calls to the browser using browser’s native support for automation. WebDriver represents a web browser, hiding specific browser details, behind the interface see figure [4.1]. Having a unified interface provides multi-browser support and allows same tests to be executed in different environments.

Selenium Remote Control (RC) is an old version of Selenium WebDriver, currently supported only in maintenance mode.

Selenium Grid allows to execute tests on different machines. Selenium Grid is useful for projects with large amount of tests or test suites that must be run in multiple environments.

Grid allows to add several physical machines to a test cluster. Grid uses the term “hub” for a central point where all tests are loaded. Hub is responsible for distributing the tests across nodes. “Node” is a remote machine with specific configuration which is attached to the hub see figure [4.2]. Nodes are totally separated from each other and may have different operating systems and browsers. Hub “decides” for

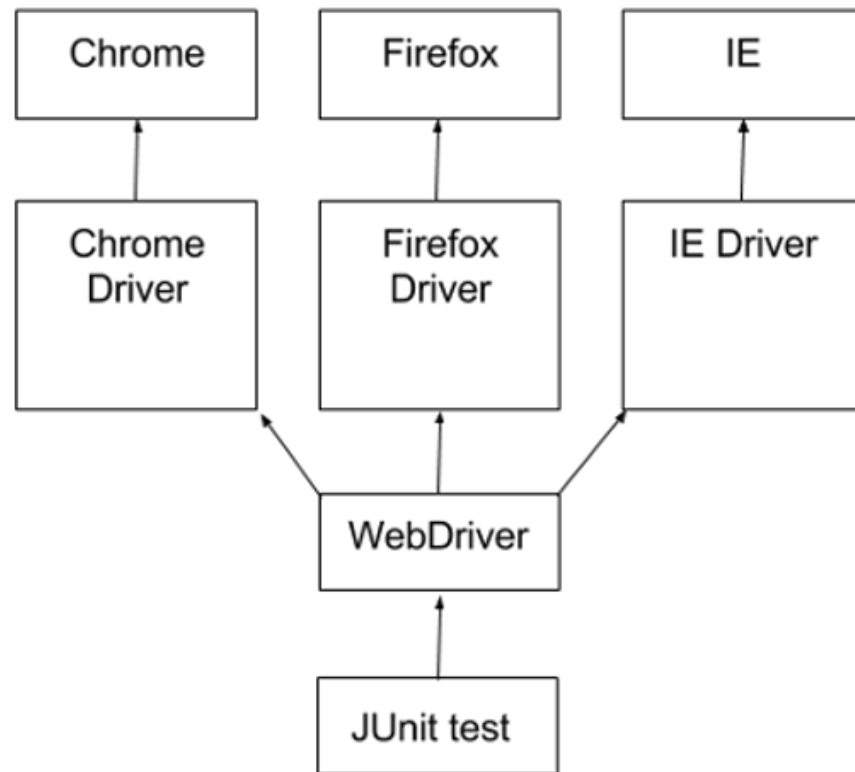


Figure 4.1 Web structure

each test suite on which node it should be executed based on test's configuration. By default every node starts eleven browsers : five Firefox, five Chrome and one Internet Explorer.

Selenium Grid is published as a separate jar file, so to setup a test hub you only need to have JRE(Java Runtime Environment) installed. For starting hub run selenium-server with hub parameter see [4.1](#). For starting a new node specify a “webdriver” parameter and the URL of the hub running [4.2](#)

```
1      java -jar selenium-server-standalone-2.30.0.jar -role hub
```

Program 4.1 Start hub

```
1      java -jar selenium-server-standalone-2.30.0.jar
2      -role webdriver
3      -hub http://http://192.168.1.1:4444/grid/register
4      -port 5566
```

Program 4.2 Start node

To create a test suite for the Grid configuration created above see an example [4.3](#)

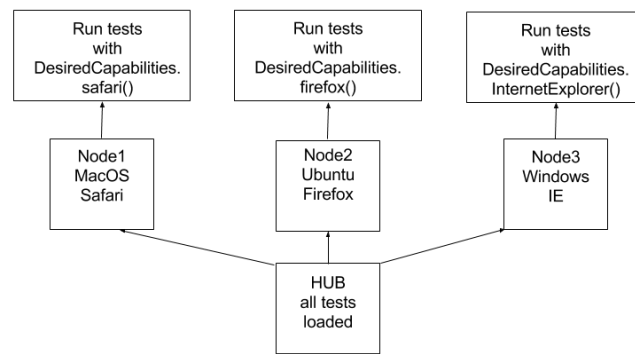



Figure 4.2 Selenium Grid structure

The basic steps are the following:

- Specify a url of tested web page line 6.
- Specify a url of a hub line 7. 
- Specify settings of the test, for example a browser line 8.
- Create an instance of remote webDriver line 9.
- Open a tested web page line 14.
- Compare actual value on a webpage with expected value line 15.
- Close web page line 20.

```

1 public class TestExample throws MalformedURLException {
2     WebDriver driver;
3     String UIUrl,nodeURL;
4     @BeforeTest public void setUp() {
5         UIUrl="http://app.example.com/hellopage";
6         hubURL="http://192.168.1.2:5566/wd/hub";
7         DesiredCapabilities capability=DesiredCapabilities.firefox();
8         driver=new RemoteWebDriver(new URL(hubURL),capability);
  
```

```
9     }
10
11     @Test
12     public void test1() {
13         driver.get(UIUrl);
14         Assert.assertEquals("Welcome", driver.getTitle());
15     }
16
17     @AfterTest
18     public void afterTest() {
19         driver.quit();
20     }
21 }
```

***Program 4.3** Selenium test example*

As it shown above setting up the basic configuration of Selenium Grid is fairly easy. After you have opened the tested Web page with “driver.get(UIUrl)”, you can use “Find Element” or “Find Elements” methods with “By” query object for locating elements on the page. For example to find an element with a class “profile” you need to use By.className query object see example [4.4](#). Selenium supports searching elements by id, tag, class, Xpath and all implications from section [3.4](#) applies to it. There is no ideal strategy for searching the required element, a developer decides which one to choose based on requirements.

```
1 WebElement avatarElement =
2     driver.findElement(By.className("profile"));
```

***Program 4.4** Search element by class*

5. REASON FOR DEVELOPING TESTBENCH



As shown in chapter [4](#) Selenium is a powerful tool for Web testing. However when using Selenium to test Vaadin application we will run into several problems. Due to these reasons a test tool for Vaadin called [TestBench](#) was created. [TestBench](#) is based on Selenium WebDriver, it solves problems with client server communication, and brings more suitable API for working with Vaadin components

5.1 Client server communication

Selenium does not know about Vaadin specific features, like client-side communication. Vaadin is a stateful framework, an event on a client side may affect a state of the application on a server side. This brings additional complexity in testing, because client-side and server-side states should be synchronized.

When event happens on the client side it will notify the server side. If this event affects the server side state, the server side will notify the client side about this change. Because of a network delay or long time code execution on the server side, there might be a delay between client side action and the change on a client. In these circumstances the client-side should wait for server side code to execute, because it might affect the next client side instruction. To handle this situation we need to add a delay in a Selenium test, see line 20 in listing [4](#) in appendix [.1](#). Rely on a delay in a test is a bad practise, because small delay might be not enough, adding a big delay on the contrary, will increase the test time execution.

5.2 Extra code

Vaadin has a rich collection of UI components, but Selenium provides only basic methods like “click” or “sendKeys”, which [requires](#) developers to [write](#) extra code for operating on Vaadin components. Same test for a text field written with Selenium [4](#) and Vaadin TestBench [5](#) takes 67 and 31 lines of code respectively.

6. DEVELOPMENT OF TESTBENCH

The estimated time for developing Testbench4 was from two months to six weeks. Our team had four members:

- Anthony Guerreiro - developer.
- Dmitrii Rogozin - developer.
- Jonatan Kronqvist - tech lead.
- Mika Mutajarvi - developer.

This is a short period of time and to manage delivering a good-quality product you have to minimize overhead costs. We believe that a team should choose tools and methodology which suits its purposes. Our team decided to use Scrum and Test Driven Development(TDD) for managing product development and try to be agile and flexible. We decided to have two week sprints.

6.1 Scrum



Scrum is a management and control process that cuts through complexity to focus on building software that meets business needs. Management and teams are able to get their hands around the requirements and technologies, never let go, and deliver working software, incrementally and empirically. The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master.

Product Owner(PO) - decides what features should the product have to maximally increase the satisfaction of the end user of the product and puts this features to the backlog. Backlog is a set of features in priority order.

Development team - is a set of professionals that are working on implementing features of the product. The team size should be from three to eight people. The team should work only on tasks from the backlog.

Scrum master - a person who should help the team to increase their productivity by enhancing the understanding of teams strengths and weaknesses.

The main idea of scrum is that development is done in short-time periods called sprints. Each spring consists of several phases:

- Sprint planning - when team decides what tasks should be done during the sprint .
- Main phase - when actual development is done.
- Sprint review - when team shows the results to the product owner.
- Retrospective - when team discuss what can be improved.

Sprint may take from one to four weeks. The development team should decide what sprint length suits their needs. During sprint planning the team chooses which tasks will be moved from a product backlog to a sprint backlog. One of the restrictions is that the task in sprint backlog should be done in one sprint. If the team thinks that the task can not be finished in one sprint this task should be divided into several subtasks.

Having such one-sprint tasks helps the scrum team to keep track of the progress easily and gives an opportunity to receive feedback for each completed task at the end of the sprint. This helps to detect problems at the early stages, when the errors does not have a tremendous feedback. Even if a feature was misunderstood by the development team and the team has to redone it completely , the team wastes time equal to the length of the spring at maximum. While in a classical waterfall model, a sequential design process in which progress is seen as flowing steadily through the phases of all development stages, the error might be found much more lately, which will have a bigger negative impact.

Another feature of Scrum is self organization of the team. The team should decide by itself which tool set to use. Tasks in scrum are not assigned to developers by a manager, but instead developers take items from the backlog by themselves. This

approach saves time and reduces stress, because a person can pick a task, which he likes and understands. Developers pick tasks that they can finish before the end of the sprint.

In our case we were not developing a new product, but releasing a new version. We did not find any arguments to change the tools  were used in the previous release we describe them in section [6.3](#).



6.2 Test Driven Development



TDD is a very popular methodology of a software development. The main idea is to write tests first and then code. The main benefits of such approach are:

- The developer is sure that his code works as intended, because all his code is tested.
- The errors are found at early stage of the development cycle, which reduces the cost of fixing problems.

Three laws of TDD [\[9\]](#) pp122|[Book page 122]

- You may not write production code until you have written a failing unit test.
- You may not write more of a unit test than is sufficient to fail.
- You may not write more production code than is sufficient to pass the currently failing test.

6.3 Tools used

6.3.1 Maven

Maven is a java-based software project management and comprehension tool. Maven is based around the central concept of a build life cycle. This means that the process for building and distributing a particular project(artifact) is clearly defined. There

are three built-in lifecycles: default, clean and site. Users can define their own life cycle.

Life cycles consist of phases. The default life cycle includes the following phases:

- validate - validates the project is correct and all necessary information is available.
- compile - compiles the source code of the project.
- test - tests the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- package - takes the compiled code and packages it in its distributable format, such as JAR.
- integration-test - processes and deploys the package if necessary into an environment where integration tests can be run.
- verify - runs any checks to verify the package is valid and meets quality criteria
- install - installs the package into the local repository, for use as a dependency in other projects locally.
- deploy - copies the final package to the remote repository for sharing with other developers and projects.

The life cycle phases are executed sequentially. For example running maven deploy executes all the previous phases (validate, compile, test...).

All maven configurations are specified in the Project Object Model (POM) file. POM is an XML file that contains information about the project and configuration details used by Maven to build the project.

Maven reduces the complexity of developing and maintaining big projects. Nowadays applications may depend on dozens of third-party libraries and frameworks. Managing those dependencies manually is very time consuming. Maven finds and downloads the exact version of the library and adds it to the project.

Maven profiles allow to have different configurations of your application for development and production or testing. All the maven configuration are in the same POM


file, that is why editing and sharing configurations between members of the team is very easy.

Finally, you have a set of predefined configurations for your application for the whole team and any developer can checkout POM file from the repository call “mvn deploy” and he will have the same version of the application with all the specified parameters and downloaded dependencies. If you updated your dependencies or fixed an error, all your team members have to just checkout the new version of a POM file.

6.3.2 Trac

Trac is an enhanced wiki and issue tracking system for software development projects. Trac may include several projects. Users or developers can create tasks (also called tickets) for these projects.

Before development a new release a product owner goes through the list of the tickets and add them to a new milestone. Milestone is a plan for the next release, which includes a set of tickets.

Tickets have different value for the end user, but developers can not always assess that value by themselves. Product owner should help the development team to figure out the value of each ticket for the end user. Based on the value and time  estimation each ticket should be prioritized. Prioritizing tickets is a very important task and should be done as soon as possible, preferable before coding starts. This gives a clear vision for all members of the team what should be done.

In the Testbench4 project we used the Trac milestone as a product backlog. On the sprint planning we estimate which tasks can be completed at the end of the sprint and move them to the sprint backlog. As a sprint backlog we used a scrum board.

Scrum board is a white board, divided into several sections for example “to be done”, “in progress”, “in review”, “closed”. Paper stickers represent tickets and the person who is working on the ticket. The workflow is the following - a developer picks the ticket from the sprint backlog queue called “to be done” writes his name on the sticker and move it to the “in progress” section. After he submitted a patch to the code review he moves the sticker to another section and so on.

Looking to the scrum board gives you a brief summary of every team member tasks

and also the current sprint progress. One can also find more detailed information about tickets and the project progress in Trac.

6.3.3 Git



As a version control system we used **Git** - distributed revision control system which focuses on speed, data integrity, and support for distributed, non-linear workflows. There are two types of revision control systems :

- Client-server - such version control systems as SVN and CVS, have a a centralised model, where there's a copy of the current code on a central server, which users copy in order to work with locally. When a user makes some changes, he updates from the central version (in case other people have made changes in the meantime), solve conflicts (same part of code was changed by different people at the same time) that might have arisen, and then push their code back to the server. Afterwards other people can check it out again.
- Distributed revision control systems such as Git, are structured on a peer-to-peer basis: instead of one centralised repository. Every developer has their own repository and there is no main repository as in client-server control systems, all repositories are "equal". Though in practise developers create a "master" repository, where everyone push their own changes and pull changes made by other developers.



One of the biggest advantage of distributed systems is that repositories are synchronised by exchanging change-sets in the form of patches, in other words if you have changed two symbols, these two symbols plus some internal information - author, time, etc. On the contrary, in systems like SVN every time you pull changes from the central repository you are downloading the whole snapshot of your application sources.



Also Git lets developers to have their local history of changes and commits, but then when pushing changes to the master repository they can rebase these changes as one commit. This helps on one hand keep a local history of intermediate steps for developer, but on the other hand have only commits for completed changes or features in the main repository.

Git has a powerful set of tools including unix commands. For example to find all commits made by one person you can use log command and pipeline it to a pattern matching command like “grep”.

Git-blame command allows you to see the history of every line of your source code. If you have questions about some particular few lines of code, you can find an author of those lines and ask him a question.

Git-bisect command - is a binary search against revision graph, which helps to find the commit which introduced a bug.

6.3.4 Teamcity

Teamcity - is a Web-based build management and continuous integration tool. Teamcity allows running multiple builds and tests under different platforms and environments. Teamcity build combines maven, ant builds, git command and bash scripts.

Teamcity builds may be started automatically or manually. One option is to create a configuration to run all tests every night or to setup running tests on every git commit. Teamcity provides also build dependencies. If project A depends on a library B, Teamcity will first build library B with its dependencies and then start build project A.

During the development cycle we used four different configurations.

- Running tests on every git commit. This configuration is started when Gerrit 6.3.5 patch is submitted. Running all tests for all browsers is very time consuming and may take several hours. That is why in this configuration includes only JUnit tests and PhantomJS tests, which does not need to run the actual browser. These tests show common errors for all browsers. Running those tests gives a developer a fast feedback, if his changes caused some problems.
- Running all tests on latest commit every night. This build triggers at specific time every night, when servers load is lower than during the day. This configuration includes all the heavy tests for specific browsers. All the tests are run on Google Chrome, Mozilla Firefox and Internet Explorer 8, 9, 10 and 11. For every test suite Teamcity will run the specific browser on a test cluster.

Running such tests is very resource consuming, but provides a confidence that the application is supported by all browsers.

- Snapshot build is run every night. This build publishes the latest version of the product to a maven repository. Users can download the snapshot build with the latest version of the product, if they want to test new features, but do not want to wait for the release build.
- Release build is run when the team releases a new version of the product. This includes building all the dependencies, running all the tests, specifying the version of the product, creating release notes, making tag in the Git repository, publishing a new version to maven repository and Vaadin Web site.

6.3.5 Gerrit

Gerrit is a Web-based code collaboration tool. Gerrit allows developers to review patches made by other developers. Gerrit has a very easy system of evaluating patches:

- -2 (veto) - patch has major problems.
- -1 (disapprove) - patch has minor problems.
- +1 (approve) - no problems found.
- +2 (approve) - can be pushed to master.

The difference between +1 and +2 is that the patch can not be pushed to git repository without having +2. The reviewer can give +1 if he is not sure about his level of competence and want someone else to inspect the patch. There are might be several configurations of the review process, figure [6.1](#) shows the process used in the Testbench4 project.

Firstly, a developer submits his changes(patch) to Gerrit. Gerrit triggers the specific build in Teamcity. This build includes building the project and running tests. After this step is finished, Teamcity returns a report about the build, if there are problems the report is send to the developer and the patch is marked as -2. If all tests pass Gerrit marks the patch as ready for review and put it to the list of waiting for

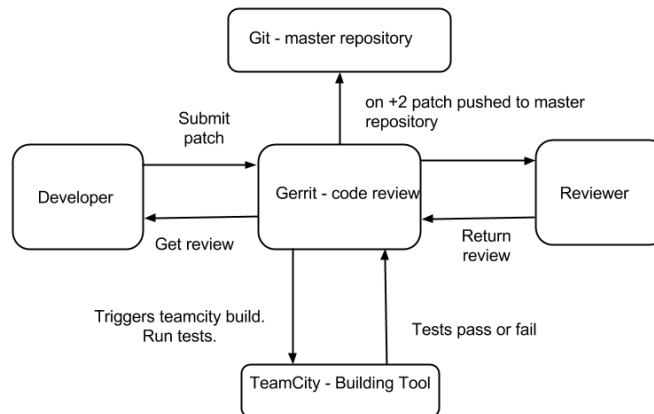


Figure 6.1 *Gerrit structure*

review patches. Afterwards the reviewer evaluates the patch. Given the patch -1 or -2 means that the developer should fix the problems, and submit the next version of the patch. The process continues until someone marks the patch as +2, meaning in can be pushed to git master repository.

Code review helps team members to follow similar code conventions, keep code clean and readable and find bugs. Also code review helps developers to know more about the whole project they are working in. Integrating Gerrit with an automated build tool, such as TeamCity, allows to run tests before publishing commit for review. The patch with failing tests is rejected automatically and an email with report for all failing tests sent to the author of the patch. As an overall code review helps to keep source code quality on a higher level.



6.4 Testbench class diagram



The hierarchy of classes in Testbench consists of many tens of classes and each class has tens of methods. Here we will describe the most important classes and the basic principles see figure [6.2](#)

TestBenchCommandExecutor handles client server communication and provides screenshot comparison implementation.

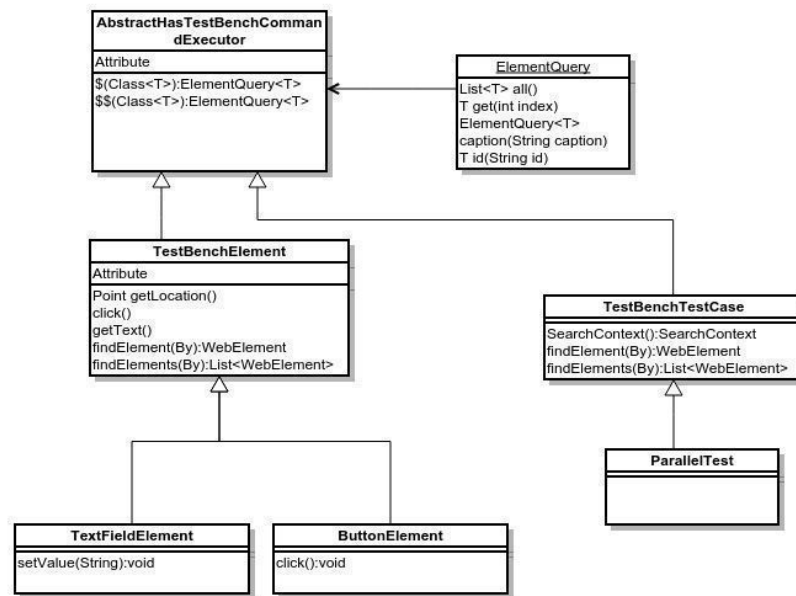


Figure 6.2 Testbench class diagram

AbstractHasTestBenchCommandExecutor class provides ‘\$(Class clazz)’ and ‘\$\$\$(Class clazz)’ methods which create a query for searching elements of the given type. ‘\$’ method builds a recursive search query and ‘\$\$\$’ a non-recursive one. Non-recursive search query looks only for direct children of the element, while it is recursive analog looks through all children of the element. Children elements are inner HTML-elements. In example [6.1](#) button and check box input elements are children elements of the div element with id id1.

```

<div id="id1">
  <input type="button">
    <input type="checkbox">
</div>

```

Program 6.1 Simple DOM example

ElementQuery<T> used for locating Web elements(Vaadin buttons, text fields, labels, etc.) on the Web page. Generic parameter T specifies the type of a searched element. ElementQuery class provides methods for searching the element based on element’s id, class, caption or other criteria. These methods can be considered as filters in the query. ElementQuery uses the Builder pattern, which helps to add several filters to build a specific query and after the query is built execute it.

Example [6.2](#) shows finding all children elements of the parent element which are

buttons:

```
1 AbstractHasTestBenchCommandExecutor elem = getParentElement();
2 List<Button> allButtons=elem.$(ButtonElement.class).all();
```

Program 6.2 Search for all buttons

To restrict search for buttons with caption “ok” we add a caption filter to the query see example [6.3](#).

```
1 AbstractHasTestBenchCommandExecutor elem = getParentElement();
2 List<Button> allButtons=elem.$(ButtonElement.class)
3   .caption("ok").all();
```

Program 6.3 Search for button with caption "ok"

TestBenchElement - is a base class for operating Vaadin components. It includes methods to access properties common to all Vaadin elements, such as "getSize()", "getLocation()", "getCssValue()", etc. TestbenchElement class uses Selenium WebElement class as a foundation and extend it's functionality by using JavascriptExecutor, which allows to execute JavaScript code, and change the default element behaviour.

TestBenchTestCase - an abstract super class of a TestBench test.

ParallelTest - supports running test in parallel threads with several browser configurations see ?? for more details.

ButtonElement, **MenuBarElement**, **TableElement**, etc. - implement specific Vaadin class features. The default naming conventions is Vaadin component name plus “Element”. In other words ButtonElement accesses buttons methods, TableElement table methods and so on.

The important aspect is that hierarchy of testbench elements is similar to Vaadin elements. That gives more flexibility when writing tests. The developer/tester can specify concrete class for getting access to specific methods of the element see example [6.4](#).

```
1 TableElement table= getElement();$(TableElement.class).first();
2 TableRowElement row=table.getRow(0);
```

Program 6.4 Test for Vaadin table

or use a more generic class to utilize method of a parent class, for example get caption of all elements, see example [6.5](#)

```

1 List<TestBenchElement> elements=
2     getElement().$(TestBenchElement.class).all();
3 List<String> captions=new ArrayList<String>();
4 for(int i=0;i<elements.size();i++) {
5     captions.add(elements.get(i).getCaption());
6 }

```

Program 6.5 Caption test for Vaadin elements

6.5 Basic test case structure



To use TestBench, the test case class should extend the TestBenchTestCase class, which provides the WebDriver and ElementQuery APIs. A developer can configure Testbench test by using following annotations:

- @Rule -defines certain TestBench parameters.
- @Before - the annotated method is executed before each test.
- @Test - annotates the tested method.
- @After - the annotated method is executed after test.

A typical test case structure is the is the following:

- Set TestBench parameters.
- Open the tested Web page URL.
- Find an element for interaction (Button, TextField).
- Interact with the elements (click buttons,menus,etc.).
- Find an element to check.
- Get and the value of the checked element.
- (optional) get screenshot.

A complete example of test UI class [2](#) and a TestBench test class [3](#) can be found in appendix A [.1](#).

6.6 Results

During Testbench4 development 41 tickets were closed the detailed information about these tickets can be found in table [1](#) in appendix B [1](#).

To improve User Experience (UX) we have tried to reduce an amount of configuration needed for using TestBench. We provide a **complete working simple test example** as a part of build-in configuration. Developers can use it as an example and extend it for their own requirements. Users may create a sample Vaadin application via GUI using Vaadin Eclipse plugin or to create a sample Test via console Maven user should use a Vaadin Maven archetype see [6.6](#).

```
mvn archetype:generate
-DarchetypeGroupId=com.vaadin
-DarchetypeArtifactId=vaadin-archetype-application
-DarchetypeVersion=LATEST
```

Program 6.6 Create Vaadin sample application command.

After beta release we made several usability tests. A Vaadin developer, without experience in using TestBench, manages to create and run a simple “button-click” test in less than 15 minutes.

Vaadin Testbench 4 is released with Commercial Vaadin Add-On License (CVAL). But if you want to look results of our work and try it out you have two options:

- Free 30-days trial period.
- One year non-commercial license. All the details how to get it are at Vaadin blog [14](#).



7. TESTBENCH VS SELENIUM

In this chapter we will summarize advantages of using TestBench against Selenium.

7.1 API built specifically for Vaadin components


Selenium operates on the DOM of the Web page and provides only basic methods of Web elements like “click” or “sendKeys”. TestBench provides a rich API for Vaadin components, which allows to operate on bigger parts of components, for example you can get a row or cell of the Table by index see [7.1](#)

```
1 TableElement table = getTableElement();
2 String value = table.getRow(0).getCell(1).getValue();
```

Program 7.1 Get Vaadin Table cell Value

7.2 Client server communication

As mentioned in section [5](#) Selenium does not handle client server communications in Vaadin application. TestBench fixes this problem.

Before calling any client side code, for example, setting a value of a text field,  TestBench waits for server side code to finish. When event happens on the client happens, TestBench sets a boolean value “hasRequestToServer” to true and sends a request to the server side. When the server side finishes its work, it returns a response and the client side sets the “hasRequestToServer” flag back to false. While there is a request to a server executing all TestBench methods on the client are suspended see figure [7.1](#)

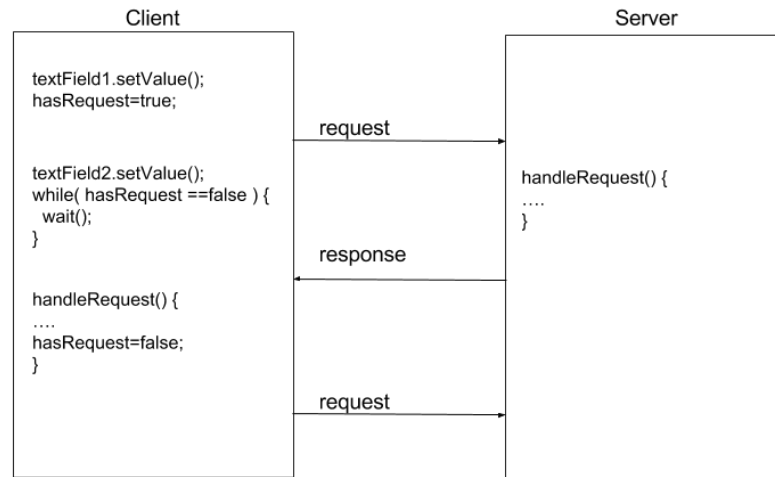


Figure 7.1 Client server synchronization in TestBench

7.3 Screenshot comparison

Since version 4.0.0 TestBench has an API for comparing screenshots. This feature was introduced to help users to test UI of the application. We believe that User Experience (UX) is very important in Web applications, and an important part of UX is look and feel of the application.

In Web applications styling is done with CSS or SCSS. The downside of CSS is that changing CSS rule for one selector may affect a lot of elements on the web page. For example change the width or margin of one element may ruin an appearance of the whole Web page.

Manual UI testing is very difficult, because of the two main challenges:

1. After some time the tester loses his concentration and does not see errors
2. Small details in applications with rich UI is hard to notice for a human. In other words if you have several text fields and buttons in different tabs or windows in the application it is hard to notice that some of them are not aligned.

Both these two problems can be solved with automatic screenshot comparison. ImageComparesment class has an overloaded compare method which takes either an

Page					
HeaderFooter ▾					
The big header					
Name					
First Name	Last Name	Gender	Birth Date	Age	Alive
Umberto	Rowling	Male	Aug 9, 1988 2:00:00 AM	29	false
Dan	Scott	Female	Sep 20, 1959 2:00:00 AM	54	false
Rita	Gordon	Female	May 16, 1992 2:00:00 AM	27	false
Marge	Schneider	Female	Dec 1, 1929 2:00:00 AM	84	false
Peter	Rowling	Female	Dec 16, 1936 2:00:00 AM	77	false
Joshua	Ross	Female	May 10, 1954 2:00:00 AM	59	true
Dan	Barke	Male	Jan 28, 1962 2:00:00 AM	51	false

Figure 7.2 Reference screenshot

Page					
HeaderFooter ▾					
The big header					
Name					
First Name	Last Name	Gender	Birth Date	Age	Alive
Umberto	Rowling	Male	Aug 9, 1988 2:00:00 AM	29	false
Dan	Scott	Female	Sep 20, 1959 2:00:00 AM	54	false
Rita	Gordon	Female	May 16, 1992 2:00:00 AM	27	false
Marge	Schneider	Female	Dec 1, 1929 2:00:00 AM	84	false
Peter	Rowling	Female	Dec 16, 1936 2:00:00 AM	77	false
Joshua	Ross	Female	May 10, 1954 2:00:00 AM	59	true
Dan	Barke	Male	Jan 28, 1962 2:00:00 AM	51	false

Figure 7.3 Screenshot with emphasized error

image or a path to the reference screenshot. The comparison is done in 16x16 blocks comparing RGB values of the every pixel in this block. If there are differences in images these parts are marked with color, so it is easier for a tester to find out what was the problem with the test see [7.2](#) and [7.3](#).



TestBench test can also be configured to automatically take screenshots of the failing tests by adding a “screenshot on failure rule” see [7.2](#). Automatically taking screenshots of failing tests helps developers to narrow the scope of a potential problem much faster.

```

1 @Rule
2 public ScreenshotOnFailureRule screenshotOnFailureRule =
3     new ScreenshotOnFailureRule(this, true);

```

Program 7.2 Adding screenshot on failure rule

7.4 Parallel testing

As mentioned in section [4](#) Selenium Grid allows to run tests on different machines with different configuration. Unfortunately Selenium does not provide a ready made solution to start those tests in parallel, the developer can use a maven surefire plugin [\[?\]](#) or JUnit “ParallelComputer” class [\[15\]](#). This requires additional work for example to use ten parallel threads for running tests using surefire plugin you need to add this XML snippet [7.3](#) to your POM file.

```

<plugins>
  [...]
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.19</version>
    <configuration>
      <parallel>methods</parallel>
      <threadCount>10</threadCount>
    </configuration>
  </plugin>
  [...]
</plugins>

```

Program 7.3 Get Vaadin Table cell Value

TestBench4 introduced a “ParallelTest” class which thread pool and execute tests in separate threads in parallel. The amount of threads can be changed by calling “Parameters.setMaxThreads()” method. Besides listing [4.3](#) shows that a developer needs to use different WebDrivers for running his test in different Web browsers or for running on Selenium Hub. TestBench uses Java annotations “@RunLocally” [7.4](#) and “RunOnHub” [7.5](#). “RunOnHub” annotation sets remote Web Driver capabilities for Chrome, Firefox and Internet Explorer 9, 10,11 by default. We believe that these adjustments in “ParallelTest” class minimize the amount of extra code and developers effort needed to setup a test environment.

```
1 @RunLocally(Browser.CHROME)
2 public class LocalTest extends ParallelTest {
3     @Test
4     public void test1() {
5         getDriver().get("http://demo.vaadin.com/dashboard/");
6     }
7 }
```

***Program 7.4** Run test in local Chrome browser locally*

```
1 @RunOnHub(http://192.168.1.2)
2 public class LocalTest extends ParallelTest {
3     @Test
4     public void test1() {
5         getDriver().get("http://demo.vaadin.com/dashboard/"); }
6 }
```

***Program 7.5** Run tests on Selenium hub on http://192.168.1.2*

8. TESTBENCH USE

In this chapter we will show several examples of using TestBench for writing automated tests and show their value for different stakeholders.

Originally TestBench was developed as a tool for writing acceptance tests for Vaadin framework it also used can be used to test any application written with Vaadin. Acceptance test determines that requirements of a specification are met. Currently (Fall 2015) there are over 6500 tests written for Vaadin framework. All tests are running in Chrome, Firefox and Internet Explorer 9,10,11 during night builds and before every release. New version of Vaadin framework can not be released even with one test failing. This strict rule helps to keep the quality of the product on a high level.

Having automated tests allow developers to refactor code without fear of breaking previous work. Developers may not know all the details of the framework and make mistakes, failing tests give sufficient information about the problem and give a confidence that new changes do not break existing code.

Having automated acceptance testing is extremely important for large open-source projects, because this reduces a cost for developers to contribute to a project. All patches to the framework are reviewed by Vaadin experts and running tests beforehand rejects fallible code.

While automated tests have great value, there are circumstances where a failing Testbench test gives a false alarm. One of the most fundamental problems of Web testing is that a developer can make a change that keeps the application completely correct, but breaks an automated test. That might be caused by changing DOM or CSS of the page, for example adding an extra div may affect searching element by xPath. Such kind of problems occurs quite often when developing new features. Using “screenshot on failure” rule [7.2](#) helps to figure out such kind of problems. If a developer get an error “can not find an element on the Web page”, but this element

Not filtered			Filtered		
filter:			filter: special		
name1	value1	property1	name5	special	foo
name2	value2	property2	name6	special	foo
name3	value3	property3			
name4	value4	property4			

Figure 8.1 Table component extension example.

is presented on a screenshot, most likely the problem is in locating the element code.

Epecially in agile development when work is done in small iterations/cycles, changes in code require changes in testing. This gives a fast feedback and an opportunity to find and fix problems early, but also brings frustration for developers that they have to fix problems both in code and tests. That might bring a false attitude that writing tests on early stages of the project, when there is no clear picture of the final product, increases the amount of work for developers.



We are sure that writing tests reduces an overall work, even if these tests have to be changed often. Usually a good rule is that every patch should add or edit at least one test suite, that also ease the reviewers job, because a test suite explains what is the reason of the patch.

In addition to having value throughout the development life cycle, Testbench tests are valuable artifacts to get end-users feedback. Because Testbench tests are executed in a browser, tests can be used for demonstrating framework or application features to the end-user.

To demonstrate the usage of Testbench we will create a test for a Vaadin table component extension. Developing Vaadin components is outside the scope of this work, we assume someone extended a Table component by adding a filter field to it. Typing value in the filter field filters values of the underlying table see figure [8.1](#).

An essential part of a Test for the filter feature is represented in listing [8.1](#).

```

1 TableElement table = getTableElement();
2 TextFieldElement filterElement=table.
3   $(TextFieldElement.class).id("filter-field").first();
4 filterElement.setValue("special");
5
6 //Comparing filtered values

```

```

7 TableRowElement row=table.getRow(1);
8 assertEquals(row.getCell(0).getValue(),"special");

```

Program 8.1 Example of table test

In the next section we will show how to improve this test by using Behaviour-Driven Development(BDD) framework.

8.1 Integrating with Behaviour-Driven Development frameworks

The main goal of BDD is to get executable specifications of a system. In other words BDD frameworks allow to write user-stories in common language, for example English, and associate them with automated acceptance tests.

Testbench can be integrated with such BDD frameworks as JBehave or Cucumber. Tests in JBehave are called scenarios see example of JBehave scenario [8.2](#)

```

Scenario: filter table contents
Given web-page with table
When typing special to filter field
Then value in row 1 and cell 0 is special

```

Program 8.2 JBehave scenario

User story steps are matched into actual Java tests using annotations. The method with an annotation interact with an application and perform the actions needed. Since TestBench tests are pure Java code and BDD steps can be run as JUnit tests, we can combine these to make JBehave run TestBench tests.

We start by extending the `TestBenchTestCase` and use JBehave's "BeforeScenario" annotation to open a tested Web page. `@Given` `@When` and `@Then` annotations are linked with corresponding steps in the user scenario. To pass parameters from the user-scenario step to a Java method "\$"- special symbol is used. JBehave implicitly casts passed value to a parameters type. The complete example of the test is in appendix A [8.1](#)

To link a Java class and a textual story file we need to create a configuration class. The simplest configuration is a one-to-one mapping between a Java class and a textual story file see example in appendix A [7](#)

So we can have both a user-story for our test explaining what should be done and a browser executed test showing the actual implementation. Picture with user story and browser implementation. We believe that user-stories scenarios greatly ease communication between stakeholders, especially if some of them do not have relevant technical background. So user-stories can be shared between stakeholders to show what is done and what is planning to be done, if there are questions executing these user-stories in a browser will help to reveal more details about it.

9. CONCLUSION

In the master thesis we have shown what are the complications about testing and testing Vaadin applications. We have described different C&R test and programmable tests, and justify that programmable tests provide several advantages.

Presenting Selenium testing framework we have shown why a Vaadin Testbench should be developed.

After six weeks of development our team Testbench 4.0.0 released in December 2014. Testbench is used both during Vaadin framework development and during development Web applications based on Vaadin framework. The biggest advantage of Testbench against other testing frameworks that it is designed specifically for testing Vaadin applications.

User case studies have shown, that a Java or Vaadin expert Vaadin, without any experience using Testbench, needs 15 minutes to setup the environment and run a simple test.

A Vaadin Testbench is a commercial product and distributed via CVAL3 license, nevertheless every one can see try it during 30 days trial period.

9.1 Discussion and Conclusion

Middle of April/ End of April

9.1.1 Summary

What has been done. What were the challenges how they were solved.

9.1.2 Advantages and disadvantages

9.1.3 Future work

BIBLIOGRAPHY

- [1] Marko Grönroos. *Book of Vaadin*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 7 edition, 2015.
- [2] Tiobe index for march 2015, 2015.
- [3] 2014.
- [4] 2014.
- [5] 2014.
- [6] 2015.
- [7] 2014.
- [8] 2014.
- [9] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.
- [10] Lei Xu and Baowen Xu. A framework for web applications testing. In *Proceedings of the 2004 International Conference on Cyberworlds*, CW '04, pages 300–305, Washington, DC, USA, 2004. IEEE Computer Society.
- [11] Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. Automating web navigation with the webvcr. *Comput. Netw.*, 33(1-6):503–517, June 2000.
- [12] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Repairing selenium test cases: An industrial case study about web page element localization. In *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, ICST '13, pages 487–488, Washington, DC, USA, 2013. IEEE Computer Society.
- [13] 2014.
- [14] 2015.
- [15] 2015.

- [16] Zhong sheng Qian, Huaikou Miao, and Hongwei Zeng. A practical web testing model for web application testing. In Kokou YÃ©tongnon, Richard Chbeir, and Albert Dipanda, editors, *SITIS*, pages 434–441. IEEE Computer Society, 2007.
- [17] Seung Hak Kuk and Hyeon Soo Kim. Automatic generation of testing environments for web applications. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02*, CSSE '08, pages 694–697, Washington, DC, USA, 2008. IEEE Computer Society.
- [18] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Improving test suites maintainability with the page object pattern: An industrial case study. In *ICSTW 2013 - 6th IEEE International Conference on Software Testing, Verification and Validation Workshops*, pages 108–113. IEEE, 2013.
- [19] Sharma M Angmo R. Performance evaluation of web based automation testing tools. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference*, pages 731–735. IEEE, 2014.
- [20] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II. Leveraging user-session data to support web application testing. *IEEE Trans. Softw. Eng.*, 31(3):187–202, March 2005.
- [21] Ondrej Kvasnovsky Jaroslav Holan. *Vaadin 7 Cookbook*. Packt Publishing Ltd, Livery Place, 35 Livery Street, Birmingham, UK, 1 edition, 2013.
- [22] 2014.
- [23] S.D. Williams. User experience design for technical communication: Expanding our notions of quality information design. In *Professional Communication Conference, 2007. IPCC 2007. IEEE International*, pages 1–13. IEEE, 2013.
- [24] 2014.
- [25] 2015.
- [26] 2014.

.1 Appendix A

```

<tr>
  <td>open</td>
  <td>
    /tutorials/selenium/selenium_record_replay.htm
  </td>
  <td>type</td>
  <td>id=number1</td>
  <td>123</td>
</tr>
<tr>
  <td>type</td>
  <td>id=number2</td>
  <td>123</td>
</tr>
<tr>
  <td>click</td>
  <td>id=add</td>
  <td></td>
</tr>
<tr>
  <td>verifyValue</td>
  <td>id=total</td>
  <td>246</td>
</tr>

```

Program 1 Generated C&R test example using Selenium IDE

```

1  @Theme("mytheme")
2  @Widgetset("com.example.testbench.MyAppWidgetset")
3  public class MyUI extends UI {
4      @Override
5      protected void init(VaadinRequest vaadinRequest) {
6          final VerticalLayout layout = new VerticalLayout();
7          layout.setMargin(true);
8          setContent(layout);
9
10         Button button = new Button("Click Me");
11         button.addClickListener(new Button.ClickListener() {
12             @Override
13             public void buttonClick(ClickEvent event)
14                 layout.addComponent(new Label("Thank you for
15                 clicking")); }
16         });

```

```

17     layout.addComponent(button);
18
19 }
20
21 @WebServlet(urlPatterns = "/*", name = "MyUIServlet",
22     asyncSupported = true)
23 @VaadinServletConfiguration(ui = MyUI.class,
24     productionMode = false)
25 public static class MyUIServlet extends VaadinServlet {
26 }
27 }

```

Program 2 Test UI class example

```

1 public class ButtonTest extends TestBenchTestCase {
2     public static final String baseUrl =
3         "http://localhost:8080";
4
5     @Before
6     public void setUp() throws Exception {
7         // Set WebDriver
8         setDriver(new FirefoxDriver());
9     }
10
11     @After
12     public void tearDown() throws Exception {
13         getDriver().quit();
14     }
15
16     @Test
17     public void testClick() {
18         //Open URL
19         getDriver().get(baseUrl + "?restartApplication");
20         ButtonElement button = $(ButtonElement.class).first();
21         button.click();
22         LabelElement label = $(LabelElement.class).first();
23         String text = label.getText();
24         Assert.assertEquals("Thank you for clicking", text);
25     }
26 }

```

Program 3 Test Bench class example

```

1 public class AppTest extends TestCase{
2     WebDriver driver;

```

```
3 String UIUrl, nodeURL;
4 @Override @Before
5     public void setUp() {
6         UIUrl = "http://demo.vaadin.com/dashboard/";
7         driver = new FirefoxDriver();
8     }
9
10 @Test
11 public void testWithoutTestbench() {
12     driver.get(UIUrl);
13     driver.manage().timeouts().
14         implicitlyWait(5, TimeUnit.SECONDS);
15     List<WebElement> elements = driver.findElements(By
16         .className("v-button"));
17     if (elements.isEmpty()) {
18         throw new RuntimeException("No buttons found");
19     }
20     elements.get(0).click();
21     driver.findElement(By.id("dashboard-edit")).click();
22     WebElement searchField =
23         driver.findElements(By.className("v-textfield")).get(0);
24
25     searchField.clear();
26     searchField.sendKeys("New Dashboard");
27     searchField.sendKeys(Keys.TAB);
28     WebElement searchButton = findButtonByCaption("Save");
29     searchButton.click();
30     driver.manage().timeouts().
31         implicitlyWait(5, TimeUnit.SECONDS);
32     String title = driver.findElement(By.
33         id("dashboard-title")).getText();
34     Assert.assertEquals("New Dashboard", title);
35 }
36
37 public WebElement findButtonByCaption(String caption) {
38     List<WebElement> buttons = driver
39         .findElements(By.className("v-button"));
40     for (WebElement button : buttons) {
41         if (button.getText().equals(caption)) {
42             return button;
43         }
44     }
45     return null;
46 }
```



```

47
48 public WebElement findButtonByCaption(
49     WebElement parent, String caption) {
50
51     List<WebElement> buttons = parent
52         .findElements(By.className("v-button"));
53
54     for (WebElement button : buttons) {
55         if (button.getText().equals(caption)) {
56             return button;
57         }
58     }
59     return null;
60 }
61
62
63 @After
64 public void afterTest() {
65     driver.quit();
66 }
67 }

```

Program 4 Selenium test for Vaadin application

5

```

1 public class TestBenchTest extends TestBenchTestCase {
2
3     WebDriver driver;
4     String UIUrl, nodeURL;
5
6     @Before
7     public void setUp() throws Exception {
8         UIUrl = "http://demo.vaadin.com/dashboard/";
9         setDriver(new FirefoxDriver());
10    }
11
12    @Test
13    public void test1() {
14        getDriver().get(UIUrl);
15        $(ButtonElement.class).first().click();
16        $(ButtonElement.class).id("dashboard-edit").click();
17        TextFieldElement searchField =
18            $(TextFieldElement.class).first();
19        searchField.setValue("New dashboard");

```

```

20         $(ButtonElement.class).caption("Save").first().click();
21
22         String title = $(LabelElement.class).
23             id("dashboard-title").getText();
24         Assert.assertEquals("New dashboard", title);
25     }
26
27     @After
28     public void afterTest() {
29         // driver.quit();
30     }
31 }

```

Program 5 TestBench test

```

1 public class FilterTableSteps extends TestBenchTestCase {
2     TableTestUI page;
3     @BeforeScenario
4         //open web page
5     public void beforeScenario() {
6         setDriver(TestBench.createDriver(
7             new FirefoxDriver()));
8         getDriver().get("http://localhost:8080");
9     }
10
11     @AfterScenario
12     public void afterScenario() {
13         getDriver().quit();
14     }
15
16     @Given("web-page with table")
17     public void theFrontPage() throws Throwable {
18         page = PageFactory.initElements(
19             getDriver(), TableTestUI.class);
20     }
21
22     @When("typing $value to filter field")
23     public void filterTable(String value)
24         throws Throwable {
25         TableElement table =
26             page.$(TableElement.class).first();
27         TextFieldElement filterElement=
28             table.$(TextFieldElement.class).
29             id("filter-field").first();
30         filterElement.setValue("special");

```

```

31     }
32
33
34     @Then("value in row $rowNumber
35         and cell $cellNumber is $expectedValue")
36     public void checkValueInCell(int rowNumber,
37         int cellNumber,String expectedValue) throws Throwable {
38         TableElement table = page.$(TableElement.class).first();
39         TableRowElement row=table.getRow(rowNumber);
40         assertEquals(row.getCell(cellNumber).
41             getValue(),expectedValue);
42     }
43 }

```

Program 6 JBehave test example

```

1  public class SimpleConfig extends JUnitStory {
2      //Specify the configuration, starting from default
3      //MostUsefulConfiguration, // and changing only what is needed
4
5      @Override
6      public Configuration configuration() {
7          return new MostUsefulConfiguration()
8              // where to find the stories
9              .useStoryLoader(new LoadFromClasspath(this.getClass()))
10             // CONSOLE and TXT reporting
11             .useStoryReporterBuilder(new StoryReporterBuilder()
12                 .withDefaultFormats()
13                 .withFormats(Format.CONSOLE, Format.TXT));
14     }
15
16     // Specify the steps classes
17     @Override
18     public InjectableStepsFactory stepsFactory() {
19         // varargs, can have more than one step class
20         return new InstanceStepsFactory(configuration(),
21             new FilterTableSteps());
22     }
23 }

```

Program 7 JBehave configuration example

.1 Appendix B

Ticket number	Description
15092	Add getValue/SetValue for PopupDateFieldElement.
14405	Remove getValue/SetValue from AbstractFieldElement class.
15088	Change API for TableElement.
15097	Remove the Getting Started PDF.
15102	Fix the driver instantiation before license check issue.
15032	Changes to the API of MenuBarElement.
15091	Change BrowserUtil class.
15089	Accordion/TabSheetElement now throw NoSuchElementException.
15085	Remove getElementInCell from TableRowElement class.
14921	Update the JavaDoc of BrowserConfiguration.
15086	Change api for TwinColSelectElement.
15087	Change api for NotificationElement
15093	Modify API for NativeSelectElement.
14918	Add setValue method to OptionGroupElement.
14919	Make TwinColSelectElement.init() protected.
14920	MenuBarElement now checks for Vaadin version 7.3.4.
14438	Update JavaDoc for scroll/scrollLeft).
14163	Fix compare screen javadoc.
13606	Replace licensing.txt with license.html
14403	Add TableHeaderElement class .
14875	Change TextFieldElement.setValue() to send Keys.TAB.
13773	Add getRow() and toggleExpanded() for TreeTableElement.
14516	Fix nativeSelect setValue and selectByText tests for phantomJS.
14385	Add contextClick for TableElement.
14889	Fix the standalone package to contain all classes
14434	TabSheetElement now works with tabs without a caption .
14426	Fix javadoc for first() and get() in ElementQuery class .
13770	Add getRow() for TableElement.
14384	Add contextClick() and doubleClick() for TestBenchElement.
14356	Fix getText() method of NotificationElement.
14486	Fix selectByText() for ComboBoxElement.
14313	Change notification close element.
14778	Update the PhantomJS Driver dependency.
14068	Add readOnly method to all vaadinElement classes()
14808	Exists() now returns false when the search fails.
13826	Fix scroll() and scrollLeft() for TableElement.
14819	Add scroll and scrollLeft for PanelElement.
13768	Add API to notification element
13769	Add getHandler to SliderElement
14372	Fix get popup suggestions in a ComboBoxElement
14790	@BrowserConfiguration methods must no longer be static

Table 1 List of closed tickets.