

Chapter

9

Testing Web Applications

CHAPTER AUTHORS

Ang Jin Juan Gabriel

Chen Shenglong Bryan

Chua Peng Chin Benson

Lian Wenhui, Florine

CONTENTS	
1	Introduction 5
1.1	Validation – Making sure you know what to build 5
1.1.1	Requirements Analysis 5
1.1.2	Behaviour Driven Development (BDD)..... 6
1.2	Verification– Making sure you’ve got what you want 7
1.2.1	Test Driven Development (TDD) 8
1.2.2	Test automation..... 9
2	Web UI Testing Using Selenium..... 10
2.1	Selenium IDE 11
2.2	Selenium RC (Remote Control)..... 12
2.3	Selenium Grid..... 13
3	References..... 14

1 INTRODUCTION

Functional correctness refers to the correctness, based on predefined specifications, of the functional qualities of an application. Functional qualities of a program are the parts that make the program useful, such the ability of a calculator software to display the correct result of some mathematical expression. In contrast, non-functional qualities are usually some form of constraints or restriction that are associated with usability or user experience features, such as the arrangement of digit buttons on the calculator.

There are **two** main parts in the process of ensuring functional correctness.

- *Validation* of requirements – evaluate software to determine whether it satisfy the requirements either during or end of development process.
- *Verification* of the correctness of the application – the process of evaluating software requirements at the start of the design phase.

1.1 Validation – Making sure you know what to build

Before we can start to program something, we need to make sure that we have a thorough understanding of the requirements – that is, the purpose and the goal of the code we are about to write. This part is extremely important, as it is very costly to rewrite entire parts of an application because it was not what the customer wanted in the first place. This step of ensuring that the purpose and goal of the program is clear and unambiguous is called validation.

There are a few ways in which requirements can be validated. We will briefly introduce two common ways to do this. The first way is to carry out **requirements analysis**, while the other is the paradigm of **Behaviour Driven Development (BDD)**. BDD, in particular, offers us a structured way to make sure that we know what we are coding for.

This portion of the chapter will introduce some of the main methods for requirements analysis (adapted from [1]) and also show how BDD can be implemented.

1.1.1 Requirements Analysis

Requirements Analysis in software engineering encompasses the tasks that go into determining the needs or conditions to meet for a new or altered product. This also takes into account the possibility of conflicting requirements from the various stakeholders, such as beneficiaries or users.

Interviews

Interviewing potential stakeholders and domain experts can give us useful information about a problem domain. Interviews are a good way of getting users to explore the problem domain with the guidance of the development team, and provide opportunities for developers to meet with stakeholders.

Focus groups

Focus groups are a kind of informal interview that is conducted as a group. Interaction and exploration are encouraged in a focus group, and participants are asked about their understanding of a specific issue or a process. Usually, there is also a facilitator who will guide the discussion and try to encourage different views to be shared and discussed.

Surveys

Surveys can be used to get responses and opinions from a large number of users in order to get a better understanding of the market sentiment towards current systems or new innovations. Surveys are quantitative in nature and are good for discovering trends and patterns.

Prototyping

A prototype is a mock up that is a scaled down version of a system. Prototypes are constructed with an aim to get users' feedback, or to validate a technical concept (a "proof-of-concept" prototype), or to give a preview of what is to come. They can also be used to compare different ideas on a small scale before committing fully to one idea. Prototyping is also used for early field-testing in order to validate basic requirements.

Existing documentation

Existing documentation is a good source of background information on existing software or user processes. It also does not require as much resources as other requirements gathering techniques.

User acceptance testing

Acceptance tests assure the customer that the system does what it is intended to. User acceptance testing is important because a system developed in a test environment (which is a simulation of the intended runtime or 'live' environment) might not work properly in the 'live' environment due to subtle differences between the two.

1.1.2 Behaviour Driven Development (BDD)

BDD focuses on obtaining a clear understanding of the desired software behaviour through discussion with stakeholders. Behaviour-driven developers use technical language in combination with the language used in the problem domain to describe the purpose and benefit of their code. This allows the developers to focus on *why* the code should be created, rather than the technical details. Doing this minimises translation between the technical language in which the code is written and the domain language spoken by the users, stakeholders, project management, etc. [2]



While Cucumber can be thought of as a "testing" tool, the intent of the tool is to support BDD. This means that the tests, which are actually usage scenarios, are typically written before anything else and verified by business analysts, domain experts and other non-technical stakeholders [3]. The production code is then written to fulfil these scenarios (see Figure 1 below).

Cucumber uses a language known as Gherkin¹ which allows for the description of the software's behavior before implementing it.

1:	Feature:	Some terse yet descriptive text of what is desired
2:		In order to realize a named business value
3:		As an explicit system actor
4:		I want to gain some beneficial outcome which furthers the goal
5:		
6:	Scenario:	Some determinable business situation
7:		Given some precondition
8:		And some other precondition
9:		When some action by the actor
10:		And some other action
11:		And yet another action
12:		Then some testable outcome is achieved

¹ <https://github.com/aslakhellesoy/cucumber/wiki/Gherkin>

```

13:      And something else we can check happens too
14:
15:  Scenario: A different situation
16:      ...

```

Line starting with keyword **Feature** followed by free indented text starts a feature. A feature usually contains a list of scenarios. You can write whatever you want up until the first scenario, which starts with the word **Scenario**.

```

Feature: Serve coffee
  In order to earn money
  Customers should be able to
  buy coffee at all times

  Scenario: Buy last coffee
    Given there are 1 coffees left in the machine
    And I have deposited 1$
    When I press the coffee button
    Then I should be served a coffee

```

Figure 1 : Example feature

After running Cucumber with all the scenarios, it will verify how many steps had passed at the end of the program.

It is able to do this as the code written to pass the test consists of a step definition which consists of the keyword, a string or a regular expression and a block of code.

For example, the following code, written in Ruby (Cucumber's natively supported language)

```

# features/step_definitions/coffee_steps.rb

Then "I should be served coffee" do
  @machine.dispensed_drink.should == "coffee"
end

```

"Then" is the keyword, "I should be served coffee" is the string and `@machine.dispensed_drink.should == "coffee"` is the block of code which is run when Cucumber matches the keyword and string.

The string can also be replaced with a regular expression, thereby matching a part of the input step as an argument to the step definition.

```

# features/step_definitions/coffee_steps.rb

Given /there are (\d+) coffees left in the machine/ do |n|
  @machine = Machine.new(n.to_i)
end

```

`(\d+)` is a regular expression term which means a sequence of digits. Its value is now stored in `n` as a string and `to_i` is called on it to convert it to an integer to be passed to `Machine.new` as an argument.

1.2 Verification– Making sure you've got what you want

The second part to ensuring functional correctness is to make sure that your program behaves according to what it is supposed to do. Note that this is linked to the previous discussion on validation as what the program is "supposed to do" is provided by validation techniques.

In order to easily test and verify the correctness of applications, it may be more convenient to start planning and structuring the project in a way that allows for the ease in executing tests. One such way is to use a paradigm known as **test-driven development**.

1.2.1 Test Driven Development (TDD)

TDD is a style of development where:

- i) An exhaustive suite of tests are maintained
- ii) No functional code (i.e., the code that implement functions of the software) is written unless it has associated tests
- iii) The tests are written first
- iv) The tests determines what the functional code should do

TDD uses a “test first” approach in which test cases are written before code is written. These test cases are written one-at-a-time and followed immediately by the generation of code required to get the test case to pass. Software development becomes a series of very short iterations in which test cases drive the creation of software and ultimately the design of the program.

Under TDD, all test cases are automated with the help of a Unit Testing Framework (UTF). The UTF assists developers in creating, executing and managing test cases (UTFs exist for almost every development environment, an indication of the popularity of this technique.). All test cases must be automated because they must be executed frequently in the TDD process with every small change to the code [9].

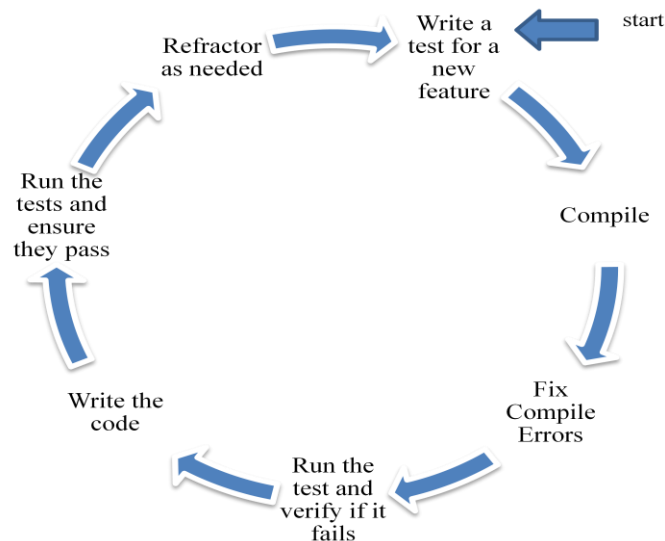


Figure 2 : TDD Cycle

TDD might seem very similar to Behaviour Driven Development. The fundamental difference is that BDD uses the word "should" instead of "test". It turns out that when you say "test", most people start thinking about what their code does, and how they can test it. With BDD, we consider - and question - what our code *should* do (verification).

As a brief example, many people who are new to TDD start pinning their code down so that nobody can break it, this is because as long as the code that they have written fulfil the requirements of the test cases, it doesn't matter how it is written. In the BDD context, however, developers tend to provide examples which demonstrate the value of their code so that others

can change their code safely. This is due to the fact that the code written has to match every step specified by the BDD's story [1].

BDD is a different way of thinking about testing, very much focused on learning, deliberately discovering areas where we perhaps thought we knew what we were doing but didn't, uncovering and helping us to resolve ignorance and misunderstanding.

1.2.2 Test automation

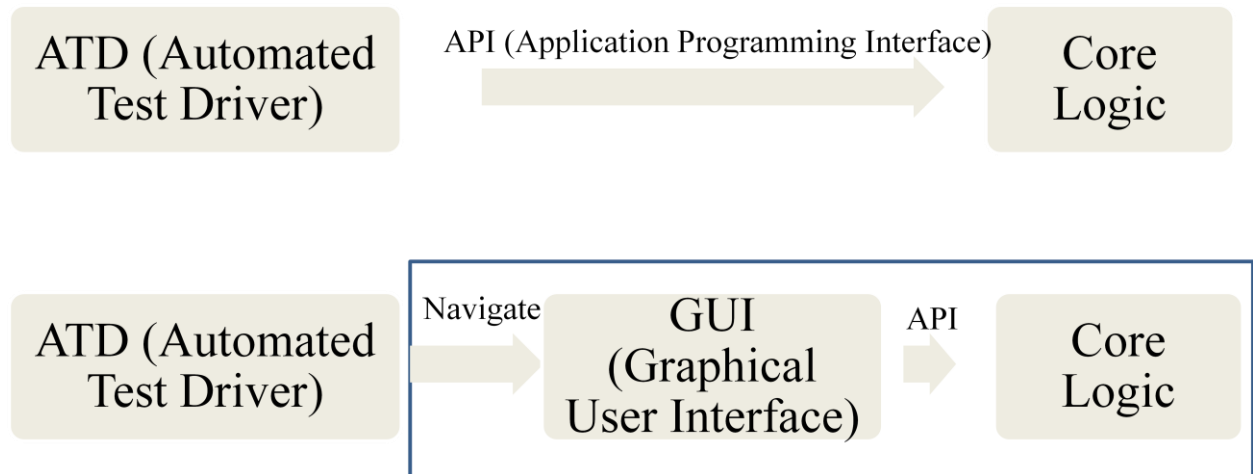


Figure 3 : API vs. GUI

Testing the API involves the automated test driver (ATD) accessing the core logic of the program directly. This may be faster, more effective and simpler to implement as it does not need to navigate the Graphical User Interface (GUI) and access program functions using the GUI. API testing can often be performed by unit testing frameworks, since it normally involves the checking of the system state and return values of the core logic.

However, API testing does not test the UI and therefore a separate testing solution may be required just for the user interface. Depending on the amount of leverage given to the testing team, the API available to use for testing the system may be limited as well. Also, since the actual use of the program involves the usage of the user interface, it may be irrelevant to just test the core logic.

With these reasons, as well as some others, such as the fact that certain applications (such as web applications) have to be tested through the GUI, has led us to focus on automating tests through the GUI.

When talking about applications, there are 3 main types, desktop, mobile and web. Most applications have a GUI and as the users of the application will be accessing the program functions through the use of GUI, it is important to ensure the functional correctness of the application through the GUI.

Automation of GUI testing allows us to simulate usage of the application automatically. Here are some tools which allow us to do that.

SWTbot

SWTbot is an open-source Java based functional/UI testing tool for testing SWT and Eclipse based applications. Because manual testing is time consuming, focusing our effort on automating the tests allows us test new features instead of retesting the old code. SWTBot's API offers enough functionality to test complex applications, is relatively easy to read and intuitive for people familiar with the basic concepts of Eclipse/Eclipse Rich Client Platform [7].

A quick 5 min demonstration on getting started with SWTbot can be found at:

<http://download.eclipse.org/technology/swtbot/docs/videos/beginners/SWTBotGettingStartedIn5Minutes/>

Abbot

Abbot provides a framework for testing GUI regardless of the current state of your code. If you are doing test-first development with lots of unit testing, then Abbot can provide the developer the tools needed to write individual unit tests. If you have an existing code base without existing unit tests, you can use Abbot's scripting to start building functional test scaffolding around your application until it is sufficiently stable to support refactoring and addition of unit tests.

In general, testing with Abbot consists of getting references to GUI components and either performing user actions on those components or making some assertion about their state.

Selenium

Due to the improving infrastructure and new focus on cloud services, web applications are gaining increased prominence. It is therefore important to know how to test these applications and hopefully automate those tests.

One of the tools which can assist us in this task is Selenium, an open-source software testing framework for web applications. It is easy to pick up due to its ability to record user actions and create test cases and it supports programming test cases in many different languages. In the next section of this book chapter, we will be focusing on Selenium.

2 WEB UI TESTING USING SELENIUM

Selenium is a set of different software tools provided with the goal of testing web application user interfaces. Each tool contributes to this goal in different ways. The Selenium suite of tools allows web application developers the ability to test their web applications using a rich and robust framework that automates many tedious operations. These operations include the locating of UI elements and comparing expected test results against actual application behaviour. One of Selenium's key features is the ability to run the same test cases on different browsers across different platforms.

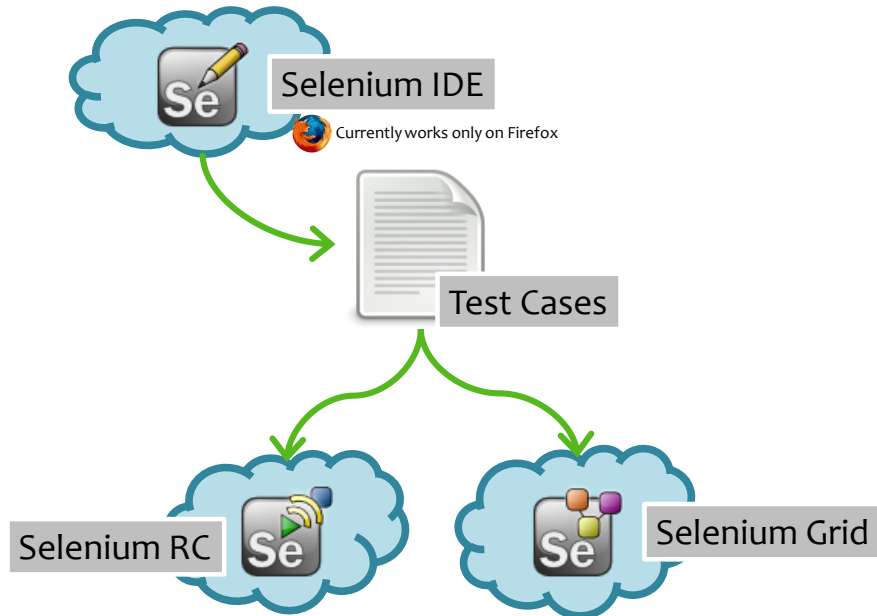


Figure 4 : Selenium Architecture

2.1 Selenium IDE

Selenium IDE (Integrated Development Environment) is a tool for building test scripts for Web application UIs. It is a Firefox plugin and provides an easy-to-use interface for developing automated tests. 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 [5].

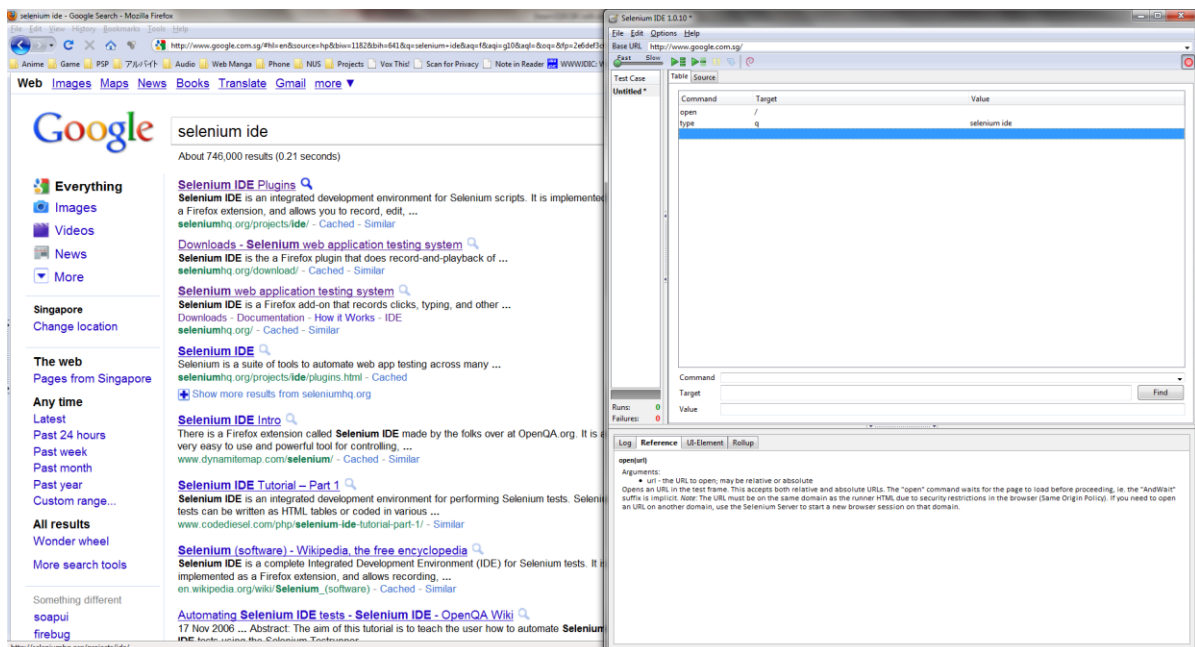


Figure 5 : Selenium IDE

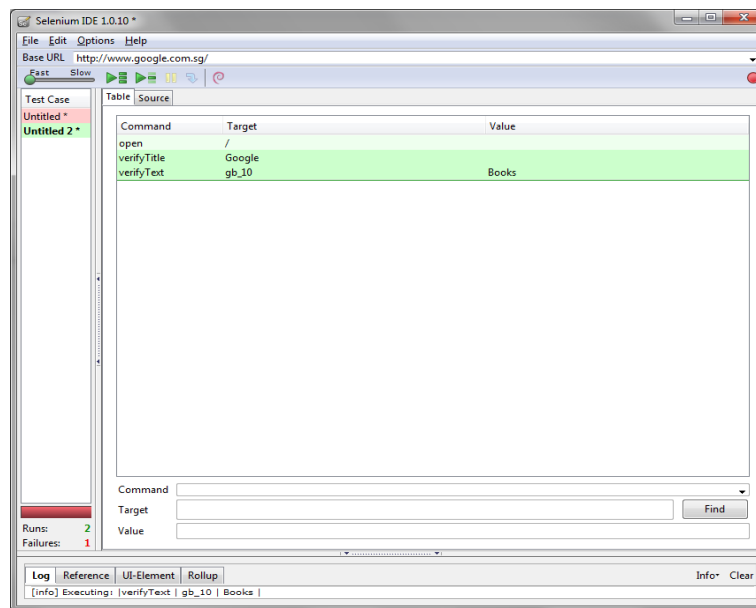


Figure 6 : Post Execution

After the execution of the test cases, the number of successful and failed test cases will be shown (see Figure 6).

2.2 Selenium RC (Remote Control)

Selenium RC (Remote Control) allows users to create more complicated test cases than with Selenium IDE alone. For example, the user is able to add conditional statements and iteration to tests.

Selenium RC (Remote Control) comes in two parts [6]:

- A server which starts an instance of different browsers (e.g. Windows Internet Explorer, Mozilla Firefox, Google Chrome) to run test cases.
- Client libraries which provides support for several languages (Java, JavaScript, Ruby, HP, Python, Perl and C#)

Browser	Argument
Internet Explorer	*iexplore
Google Chrome	*googlechrome
Mozilla Firefox	*firefox
Safari	*safari

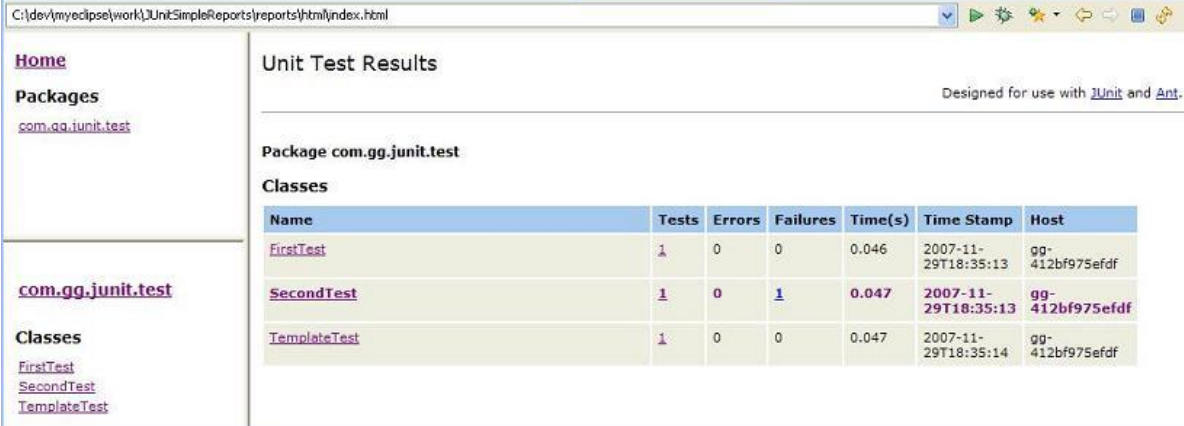
Table 1 List of Browsers

By itself, Selenium RC does not have the function to support the reporting of results. However, there are many third party reporting tools available [15].

Some common used third party reporting tools in java:

1) JUnit Report

Configure “build.xml” script in Ant and execute Selenium, this will generate a JUnit report for Selenium test.

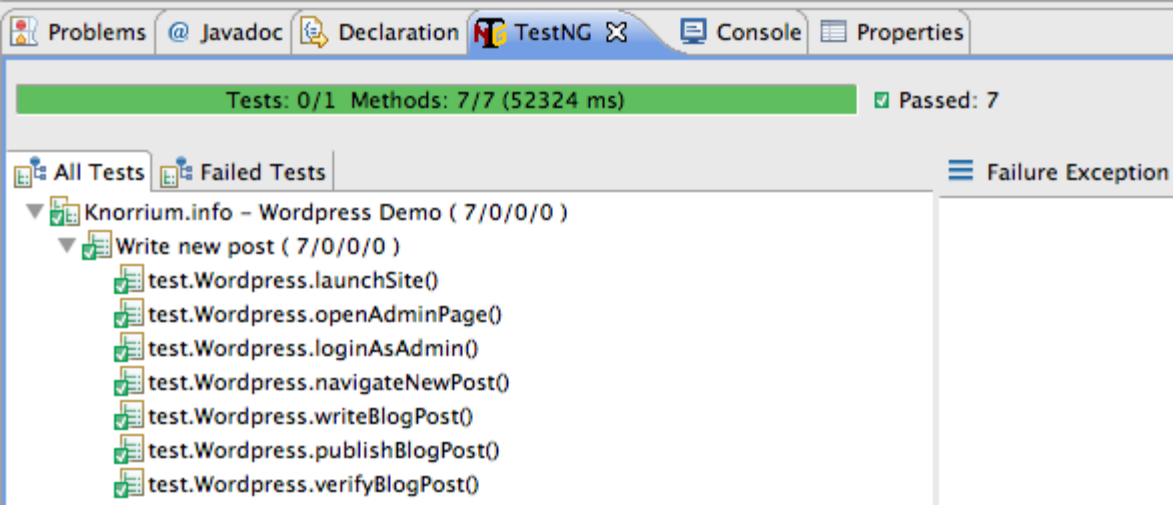


Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
FirstTest	1	0	0	0.046	2007-11-29T18:35:13	gg-412bf975efdf
SecondTest	1	0	1	0.047	2007-11-29T18:35:13	gg-412bf975efdf
TemplateTest	1	0	0	0.047	2007-11-29T18:35:14	gg-412bf975efdf

Figure 7 : JUnit Report

2) TestNG Report

TestNG framework generates an HTML report which list details of tests.



Tests: 0/1 Methods: 7/7 (52324 ms) Passed: 7

Failure Exception

- Knorrium.info - Wordpress Demo (7/0/0/0)
 - Write new post (7/0/0/0)
 - test.Wordpress.launchSite()
 - test.Wordpress.openAdminPage()
 - test.Wordpress.loginAsAdmin()
 - test.Wordpress.navigateNewPost()
 - test.Wordpress.writeBlogPost()
 - test.Wordpress.publishBlogPost()
 - test.Wordpress.verifyBlogPost()

Figure 8 : TestNG Report

2.3 Selenium Grid

Selenium Grid allows the Selenium RC solution to scale for large test suites and for test cases that must be run in multiple environments. Selenium Grid allows you to run your tests in parallel, that is, different tests can be run at the same time on different remote machines. This has two advantages.

Firstly, if you have a large test suite, or a slow-running test suite, you can boost its performance substantially by using Selenium Grid to divide your test suite to run different tests at the same time using those different machines.

Secondly, if you must run your test suite on multiple environments, Selenium Grid allows you to have different remote machines running your tests in them at the same time.

In each case Selenium Grid greatly improves the time it takes to run your suite by making use of parallel processing [5].

3 REFERENCES

- [1] BDD vs TDD, Electronic References, Retrieved March 27, 2011 from <http://stackoverflow.com/questions/3905115/is-bdd-a-replacement-of-tdd>
- [2] Behavior Driven Development. Electronic References. Retrieved March 12, 2011, from http://en.wikipedia.org/wiki/Behavior_Driven_Development
- [3] Cucumber. Electronic References. Retrieved March 20, 2011, from <https://github.com/aslakhellesoy/cucumber/wiki>
- [4] List of Selenium RC Browser Launchers. Electronic References. Retrieved March 12, 2011, from <http://stackoverflow.com/questions/2569977/list-of-selenium-rc-browser-launchers>
- [5] Selenium. Electronic References. Retrieved March 13, 2011, from <http://www.seleniumhq.com/>
- [6] Selenium RC. Electronic References. Retrieved April 9, 2011, from http://seleniumhq.org/docs/05_selenium_rc.html#reporting-results
- [7] SWTbot. Electronic References. Retrieved March 12, 2011, from https://publications.theseus.fi/bitstream/handle/10024/7470/Mazurkiewicz_Milosz.pdf?sequence=2
- [8] TDD (Test Driven Development). Electronic References. Retrieved March 13, 2011, from http://www.testingeducation.org/conference/wtst4/pjs_wtst4.pdf
- [9] Teaching Unit Testing Using Test Driven Development. Electronic References. Retrieved March 20, 2011, from <http://www.slideshare.net/Softwarecentral/teaching-unit-testing-using-testdriven-development>
- [10] TestNG and Selenium. Electronic References. Retrieved April 9, 2011, from <http://testng.org/doc/selenium.html>
- [11] Why selenium. Electronic References. Retrieved March 13, 2011, from http://www.wdvl.com/Authoring/Java/test/Avneet_Mangat03242010.html