Chapter

3

Testing

CHAPTER AUTHORS

Michael Atmadja

Zhang Shuai

Richard

PREVIOUS CONTRIBUTORS : Ang Jin Juan Gabriel; Chen Shenglong Bryan; Chua Peng Chin Benson; Lian Wenhui, Florine

CONTENTS

1 INTRODUCTION

1.1 Why do you need Quality Assurance (QA) Test?

In most companies, before complete information can be completely received by the developers involved in building the software, the information itself has changed in between. As a result, the user's requirement may not necessarily be described accurately by the business analyst. Thus, what has been perceived as correct information by developers may not be what the users want. With these discrepancies occurring very often, it is not rare for a company to receive complains once they launched the software.

Hopefully and surely, by conducting QA test, the discrepancies in the information can be minimized. Any differences between the software functionalities and its requirements can be referred back to the developers by QA team before releasing the software.

1.2 When is Quality Assurance (QA) Test done?

QA Test can be done at different stages of the development. The time to start a QA test depends heavily on the development method adopted. Different method of development put different emphasis on when a QA test should be done. For example, waterfall method allows QA test only at the end of the cycle, while in agile environment, testing is done as part of the iteration.

In waterfall method, larger changes are made all at once, this require large amount of time for the developers to complete their changes. Hence, QA team is only able to proceed in testing at the end of development cycle. While the software is under development, QA team will not be able to start working on the said software.

In agile environment, which involves short iterations of working software, QA team is expected to devise test cases on the future software functionalities while the development is ongoing. QA team has very little time to conduct their QA test in agile environment and hence, is expected to work very effectively and efficiently. Very often, in some agile methodology, the developers themselves are the QA tester. In some methodology where developers are expected to be the master of both development techniques and testing strategies, only the very experienced software developer makes up the team.

1.3 V-Model of software development process

V-Model was proposed by Paul E. Brook in 1986. It is a system development model which helps to simplify the complexity of systems under development. V-Model is said to be the modified version of Waterfall method, but represented in a V shape, hence the name.

This model is considered to be a very important model as it demonstrates the relationships between the development stages and its corresponding testing stages. There are various ways V-Model has been represented and one of the V-Model of software development can be seen at figure 1 below.

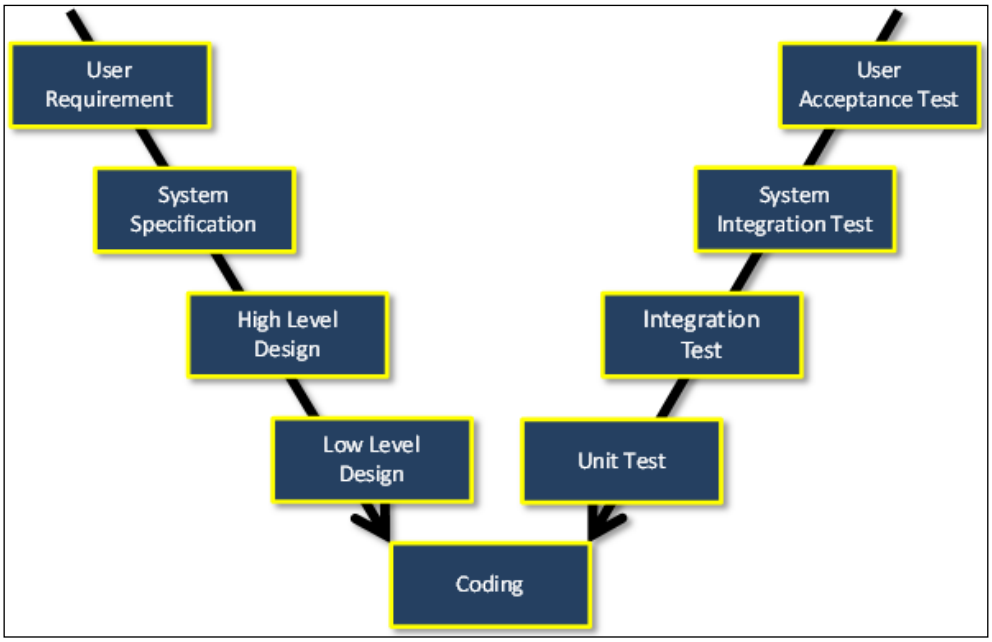


Figure 1: V-Model of software development

1.4 Stages of the V-Model

Each node in the V-Model represents a stage in the software development cycle and its corresponding testing stages. It starts from the "User Requirements" down to "Code" and eventually to "Acceptance Testing". In Waterfall method, one cycle is considered to be one cycle of development, but in Agile method, one cycle may only represent one iteration and the cycle continues to the end of the iterations.

1. **User requirement**

   In this stage, the project and its functions are defined which will be a guideline for the rest of the development process. This stage is very important because the definition has to be closely, if not perfect match of what the users requested.

2. **System specification**

   System specification deals with how the possible design of the whole project is going to be. If there are any functions which don't fit to the design, users will be notified and necessary changes will be made.

3. **High level design**

   This stage is also known as Architecture design. In this stage, the architecture of the software is designed based on the discussions of the previous two stages. The architecture is used to ensure a complexity of the software can be managed; otherwise problems related to high level of complexity will occur at the later stage of the development.

4. **Low level design**

   This stage is also known as Module design. In this stage, architectural design is broken further into sub-problems to be implemented by the team.

5. **Coding**

Developers will start coding at this stage according to the specifications defined.

6. **Unit test**

Each method is tested to ensure correctness of the result. If any of the methods fail, it may affect the end result.

7. **Integration test**

In this stage, different entities are put together and tested to find any flaw in the interfaces.

8. **System integration test**

Further test is done on the system. In this stage, the system is checked against the system specification to see if it has fulfilled the requirement.

9. **User acceptance test**

In this stage, the software is tested against the full requirement specified at the start of the project. If all requirements have been met, the product is ready to be launched. Otherwise, the tester will revert back to the developers for fixes.

In the whole process of software development, many parties are involved in making sure that the software is fit for release.

For stage (1), project manager and business analyst will discuss on the requirement based on the request from the users. In the next stage, project manager and its team will discuss on the system specification. In stage (3), a software architect (either from the team or from the company) will define an architectural design of the software. Once done, stage (4) to stage (7) will be executed by the team. Stage (8) and stage (9) is the final stage of the whole process and is done by a team of people who specialize in Quality Assurance. This team of people can be part of the development team (in Agile method) or independent from the development team (in Waterfall method).

This V-Model of software development widely represents how a company develops their system.

2 TESTING PRINCIPLES

As with any other occupation, there are some fundamental techniques and rules involved in doing QA testing. The Quality Assurance process is costly as it delays the development progress of the program. Thus, it is important to know these techniques and rules to ensure efficient and effective testing. This in turn will reduce the cost of doing QA.

In this section of the book we will discuss the techniques and rules that someone who aspires to be a good QA tester should know. We will first discuss the characteristics of a good test case and test suite. Next, we will show some of the common strategies employed during QA testing.

2.1 Characteristics of a Good Test Case

Test cases[1] are testers' most important tool in doing their job. A well designed test case could significantly improve the efficiency of testing. Similarly, a poorly thought out test case would make testing less efficient and more costly.

Good test cases share several similarities which are:

- **They have a clear purpose**
- **They focus on testing only a few aspects of the test subject[2]**
- **They produce output which can be easily verified by the tester**
- **They give reproducible errors.**

Remember, ultimately from the developer's point of view, the purpose of testing is to discover which part that needs to be fixed. Thus, not having a clear idea of which section is being tested or having too many aspects tested at the same time defeats the purpose. This is because if such test case failed, the developer would not know which part they need to fix.

It is also important to know the expected output of the test case so the tester can verify that the test subject does not contain any logic error. Most of the time, a bug is produced by logic error in the system. That is to say, the test case could execute perfectly but it gives the wrong output. If the tester does not know the expected output and assume the result given by the computer is correct, they will not be able to detect these logic errors.

In the industry, it is also important to be able to reproduce errors. As mentioned before, QA delays the progress of the development process and thus should be done as efficiently as possible. As such, each and every failed test cases reported by the tester should definitely contain an error.

In the industry, you will typically work with a complicated system and thus there is bound to be multiple points of failure[3]. You would want to make sure that any failure encountered during testing is purely caused by the test subject. One way to ensure this is by trying to reproduce the mistake.

In practice, you will have to create multiple test cases to assure the correctness of your test subject. The collections of these test cases are called test suites. As with test cases, there are some characteristics that make a good test suite. These characteristics will be discussed in the next subsection.

2.2 Characteristics of a Good Test Suite

As mentioned before, a test suite is a collection of test cases that is used to test the same test subject. When designing a test suite, it is important to bear in mind two things namely: test case independence and test coverage.

Independence

Test case independence simply means that there should not be interaction between the test cases (i.e. there shouldn't be test case which depends on other test case's output). That is to say

[1] Test cases are a set of input with known expected output that is used to test the functional correctness of the program

[2] Test subject could range from a piece of code (in unit testing) to a general functionality (in system testing)

[3] Points of failure are aspect of a particular system which could potentially cause the system to break down.

you should be able to run the test cases in any order without having any impact on the overall evaluation of the test suite.

If there is a test case that is dependent, the developer will not know for sure whether the functionalities tested are the one that causes the failure or the functionalities in the test case that it depends on is the one that needs to be fixed.

Coverage

Coverage is a very broad term in testing as it can refer to many things. When a person refers to coverage, usually they will refer to line-coverage, block-coverage, branch-coverage, or path-coverage. However, the definition can be extended to functionality coverage as well when System testing and Acceptance testing is concerned.

Line coverage measures the percentage of lines executed during the entire test compared to the entire number of lines in the test subject. Block coverage measures the percentage of blocks (codes in loops or in if statements) executed during testing as compared to the whole test subject. Branch coverage measures the number of branch visited during the testing process as compared to the actual number of branches that exist in the test subject. Similarly path coverage measures the number of path traversed during the testing process as compared to the actual number of path that exist in the subject.

If you recall, one of the characteristics of a good test case is that it only focus on a few aspect of the test subject. However, instinctually you need to ensure that all aspect of the program is sufficiently tested before released to the market. Thus, you want to create a test suite that has a good coverage statistic.

There are methods that you can employ to see if a test suite has a good coverage. There are softwares which are able to measure the coverage as you test. However, you can also do it manually using the testing matrix.

Now that you know the basic of test case and test suite, you might wonder how they are used in testing and if there are strategies in which they are applied. We will discuss them in the next section.

End of Section Note:

It is important for testers to be able to create good test cases and robust test suites. To guide you along and summarize the points raised, there are 6 questions you can ask yourself to gauge the efficiency of your tests:

For test-cases:

1. Do I know what I am testing?
2. Am I testing for too many aspects at the same time?
3. Do I know the expected output?
4. Can I reproduce the error easily if this test case fails?

For test-suites:

5. Can I rearrange the test-cases freely?
6. Have I sufficiently test the different aspects of my test subject?

3 TESTING TECHNIQUES & APPLICATION

3.1 The Testing Matrix

The testing matrix is a simple yet very useful method that allows tester to both plan their test suite as well as document them at the same time. The figure 3.1 shows how a typical test matrix looks like.

	SPEC ITEM 1	SPEC ITEM 2	SPEC ITEM 3	SPEC ITEM 4	SPEC ITEM 5	SPEC ITEM 6
Test Case 1	X					X
Test Case 2	X	X		X		X
Test Case 3			X	X		X
Test Case 4			X	X		X
Test Case 5	X					X
Test Case 6		X				X
TOTALS	3	2	2	3	0	6

Figure 3.1 Example of Testing Matrix

Constructing a Testing Matrix

A testing matrix consists of 4 elements: test-case column, test specifications row, checklist area, and the tally row as shown in the Figure3.2.

The test-case column contains the different test cases that you are using in that particular test-suite.

Test specifications row contains the different test specifications that your test subject need to have. The specifications can be flexibly determined to reflect the level of testing done. For example, in unit testing a specification can be a particular piece of code but in system testing, the specification can be a functionality that the system needs to be able to perform.

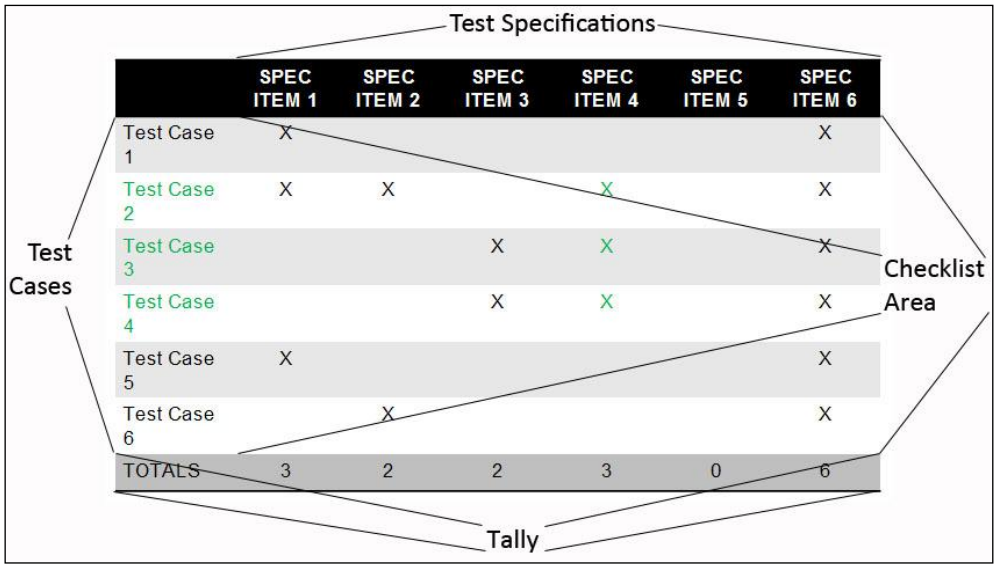


Figure 3.2 Testing Matrix parts

Using the Testing Matrix

Testing matrix can be used both ways: to document the functionality tested by a test case or to design a test case based on the functionality you want to test. To document the functionality tested, you can list down the test cases in the test suite in the first column. Subsequently, mark the columns corresponding to the test specifications that are tested for each test case.

If you have no idea where to start designing your test suite, test matrix is a good way to develop a test suite with good functionality coverage. You can mark different permutations of specifications to be tested and develop the test cases. This will make sure that the resulting test suite test each specifications and interaction between them at least once.

Once you have filled the checklist area, you should tally the number of times each test specification is tested in the test suite.

Other Advantages of Testing Matrix

Testing matrix helps you to quickly identify insufficiently tested specifications. You can use the numbers in the tally row to see if your test suite has sufficiently tested every specification.

Testing matrix helps you to determine which test cases to modify when a specification change. In case there is a change in a specification, you can check the checklist area for which test cases need to be modified.

3.2 3.2 Testing Strategies

There are different classifications of the strategies employed in software testing. The one that we will discuss in this chapter is based on how the process is conducted.

Several strategies that are based on these classifications include:

- **Regression Testing**

  Regression testing normally occurs in iterative projects. Basically, it aims to make sure that the changes introduced in the new iteration will not affects the other functionalities implemented in the previous iteration.

- **Smoke Testing**

Smoke testing can be seen as a "rule of thumb" testing. It involves checking if the system does what it is supposed to do. If the system fails to do so, it will fail the smoke testing.

- **Exploratory Testing**

Most of you would be familiar with the exploratory testing. As the name suggest, the exploratory testing is somewhat without plan. It involves trying to simulate what the user will do with the system. An example of exploratory testing would be Beta release for games.

- **Scenario Testing**

Scenario testing involves testing the system following a set of predefined scenarios that the tester created. Typically the scenario will be closely related to the Use Case documents that are created during the design analysis phase of the development. This will make sure that at least the system will not fails for general activities that it is designed to handle.

- **Guerilla Testing**

Guerilla Testing is similar to exploratory testing. However, guerilla testing involves a strict time period during which the system will be tested continuously. The tester will also try to create extreme test cases to try to break the system. Typically guerilla testing is done by experienced QA tester to ensure that the system is robust enough to handle extreme values.

- **Load Testing**

Load testing is commonly used in Enterprise systems and web applications which need to support multiple clients. It is used to ensure that the system will be able to handle and serve multiple requests from multiple clients at the same time.

These are just a small portion of the different strategies that is commonly used in the industry. However, regardless of the strategy employed the basic of testing still lies with creating an efficient test case and test suite that suit the purpose.

End of Section Note:

Testing Matrix is a simple technique to document as well as create a test suite

It's important to pick the right testing strategy that fits our intention.

4 TESTING TOOLS

Software testing tools can simplify testing process dramatically. It also allows testers to find more defects hidden in the software and achieve a better release quality in the end.

There are a lot of open source testing tools which can be downloaded for free. People may also choose to purchase commercial software testing tool which generally have more functionality to help and support the software testing process. You should testing tools that fit and add value to the existing testing process.

In this section, we will discuss four major categories of software testing tools namely: test case generation tool, test automation tool, bug tracking tool and test case management tool.

4.1 Test Case Generation Tools

As the name suggest, test case generation tool is used to generate test case.

In recent years, most of test case generation tools use the model-based approach. A model based test generator is an automated process which accepts two main inputs: a formal model of software under test and a set of test generation directives which guide the tool in its generation. The output will be a set of test cases.

Compared to generating test case manually, automatic test case generation has several benefits such as time saving and non-biasness. Test case generation is a time consuming task and it saves much resources by automating it. Human involvement in manual test case generation process may leads to the involvement of human biases.

Some examples of test case generators are TOSTER[4] and UNITESK[5]

TOSTER (The Object-oriented Software Testing Environment) is a test generation and execution system produced by the Warsaw University of Technology. It incorporates technology for mapping the information in UML diagrams to the source code of an application. It also generates and runs test cases based on expected results derived from the UML state diagrams. There appear to be two test generation algorithms, but no explicit testing directives.

The UniTesK (Unified Testing and Specification Toolkit) is invented by ISPRAS (Institute for System Programming of the Russian Academy of Sciences).The model is specified in either J@VA or C@++, which are specification languages designed for Java and C++ code respectively.

The specifications are in the form of pre- and post-conditions, and are coded as comments in the classes and methods to be tested. The test cases are generated by applying branch coverage of the specification of the post-condition, and this test directive seems to be implicit in the test generation process. Since UniTesK is close to the source code, the test drivers are created as part of the test generation process, and not in a separate abstract to concrete translation phase.

4.2 Test Automation Tools

Test automation tool is a piece of software that can conduct testing process automatically. Given an input of test cases, it is able to control the execution of test and compare actual and predicted outcomes. With test automation tools, test cases can be run quickly and repeatedly. Therefore, it saves testers from the laborious and time consuming process.

There are two types of test automation tools: Code driven testing and GUI driven testing.

Code driven testing gets rid of the graphical user interface (GUI) and tests the public interface of classes, modules and libraries directly. Therefore, it is faster, more effective and simpler to implement. Usually, code driven testing is conduct by unit testing frameworks (for example, JUnit).

GUI Testing tool on the other hand generates user interface events like keystrokes and mouse clicks and observe the resulting change. It would then validate whether the behaviour is correct or not. The advantage of this approach is that it is not necessary to have the knowledge on software development. Any application with a GUI can be tested by GUI testing tool.

Some examples of test automation tools are Selenium[6] and JUnit[7]

[4] Toster.sourceforge.net
[5] www.unitesk.com

Selenium is an open source GUI software testing framework for web applications. It provides a rich and robust framework that can perform many tedious testing operations automatically. It is a Firefox plug-in 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. Another key feature of Selenium is the ability to run the same test cases on different browsers across different platforms.

JUnit is an open source unit testing framework for the Java programming language. It is mainly used to write and run automated tests. It provides a comprehensive assertion facility to verify expected versus actual results.

4.3 Test Case Management Tools

Test Case Management (TCM) is a way of organizing test assets and artefacts such as test results and documentations. This will provide the tester with easy accessibility and usability. Some TCM allows user to keep a version control of the individual test case, combine test cases to make test scripts and track test results against executed test cases. This allows companies to save time and expenses at the same time they can reuse their test cases.

Some examples of Test Case Management Tools are TestLog and TestLodge:

TestLog[8] is commercial test case management software which allows user to create, manage test case and even an entire test plan. It also allows importing and exporting of existing test case database from and to CSV file. It is able to document both manual and automated test case. The program also provides inline reports which can provide valuable information such as the progress and helps tester to estimate if the project still sticks to the schedule.

TestLodge[9] is paid testing software. It lets user to handle their test plans, test cases, requirements and test runs easily and efficiently. It is a hosted software testing software, thus the end user gets everything set up. Thus, installation and multiple users are not the issue. TestLodge also can integrate easily with your existing tools allowing you to report and track failed test cases on your existing software.

4.4 Bug Reporting / Tracking System

A bug tracking system is an application that can be used to keep track of the reported bugs. It is also known as issue tracking system.

[6] Seleniumhq.org
[7] Junit.sourceforge.net
[8] www.testLog.com

[9]www.testlodge.com

Bug tracking system mostly consists of a database which stores the facts about reported bugs. It contains the characteristic information of each reported. The information includes the following:

- Time a bug was reported
- Status
- Severity
- Erroneous program behaviour
- Details on how to reproduce the bug
- Identity of the person who reported it
- Identity of programmers who is going to fixing it

A typical bug reporting system has two roles: Administrator and Programmer. Administrators are able to configure permissions based on bug status, change the bug status, delete bug report and assign bugs to certain programmers. Programmers can view the active bug assigned to them, inform a fixed bug to administrator and raise another bug.

Bug reporting system is very important in software development. It provides a clear, centralized overview of bugs and its facts. It allows all team members can see what bug has been found and in which area.

Bug reporting system, also allows clear coordination as the responsibility of each team member is clearly defined. This allows for better team cooperation and communication.

An example of bug reporting tools is BugZilla[10]. BugZilla is an open source bug tracking system. It allows user to get a handle on the software development process by doing the following:

- Track bugs and code changes
- Communicate with teammates
- Submit and review patches
- Manage quality assurance (QA)

It is a powerful tool that will help your team get organized and communicate effectively.

End of Section Note:

Based on their functionality, testing tools can be categorized into 4 main category:

Test Case Generation Tool

Test Automation Tool

Bugs Reporting / Bug Tracker System

Test Case Management Tool

[10] www.bugzilla.org

APPENDIX 5.A: TEST DRIVEN DEVELOPMENT

TDD is a style of development where:

i) An exhaustive suite of tests are maintained

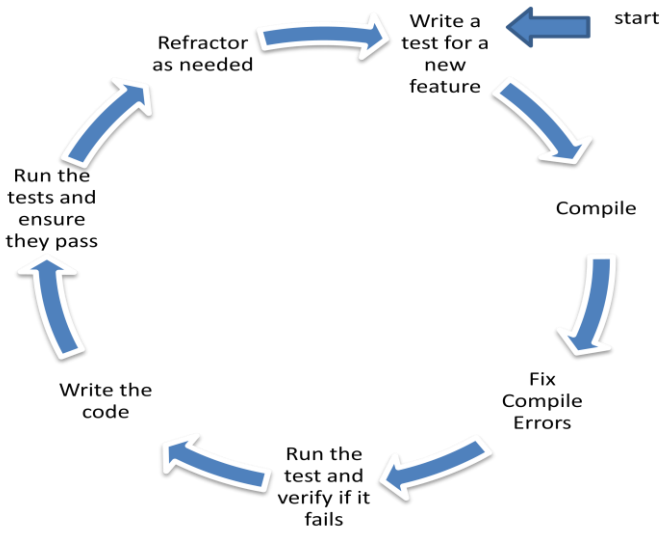
ii) No functional code (i.e., the code that implement functions of the software) is written unless it has associated tests

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.



To summarize, here are the 3 laws of TDD:

- Do not code unless there is a failing test case
- Do not write more test than sufficient to fail, compilation failures are failures
- Do not write more code than sufficient to pass the failing test

APPENDIX 5.B: 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.

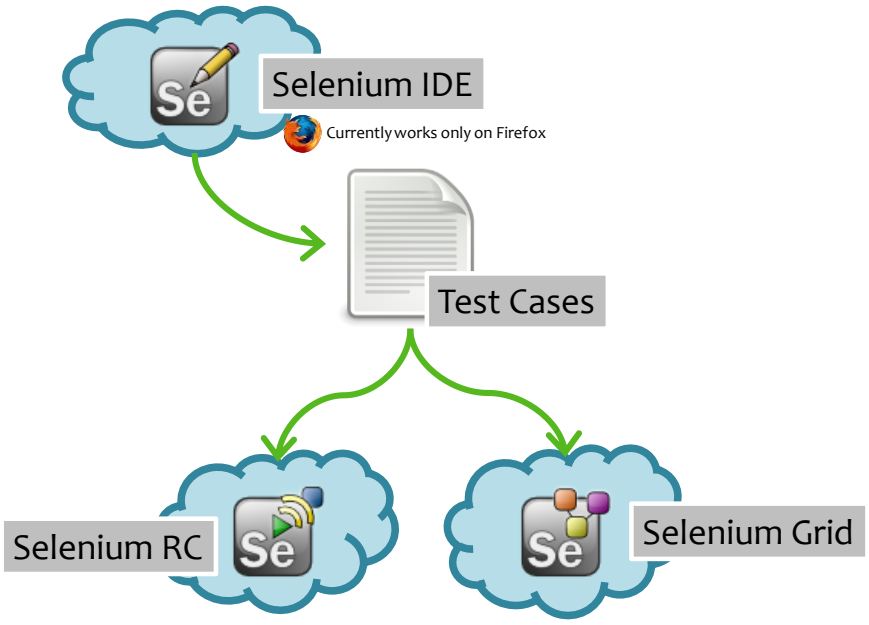


Figure 1 : Selenium Architecture

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.

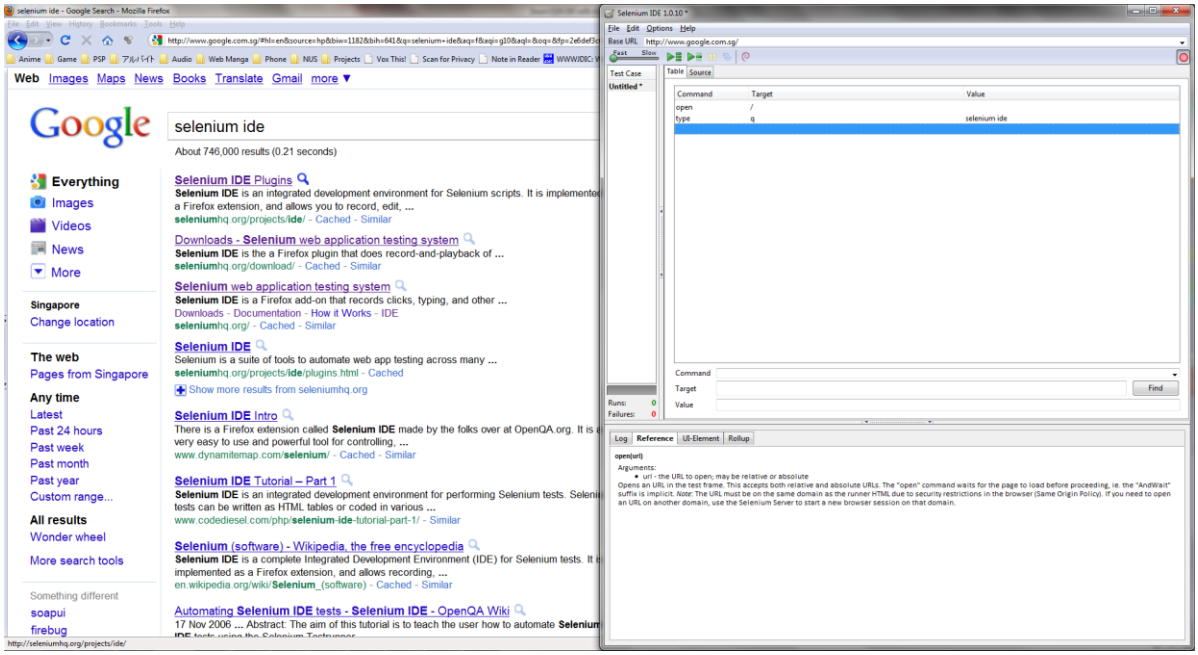


Figure 2 : Selenium IDE

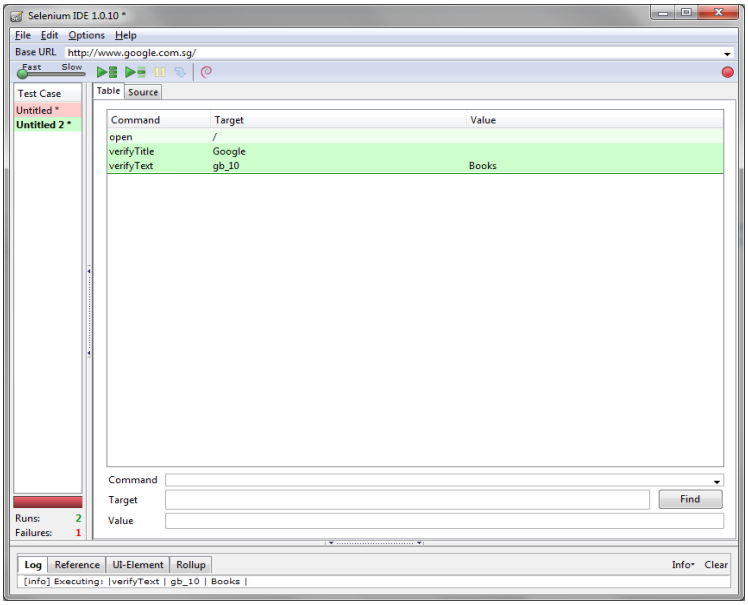


Figure 3 : Post Execution

After the execution of the test cases, the number of successful and failed test cases will be shown (see Figure 3).

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:

- 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.

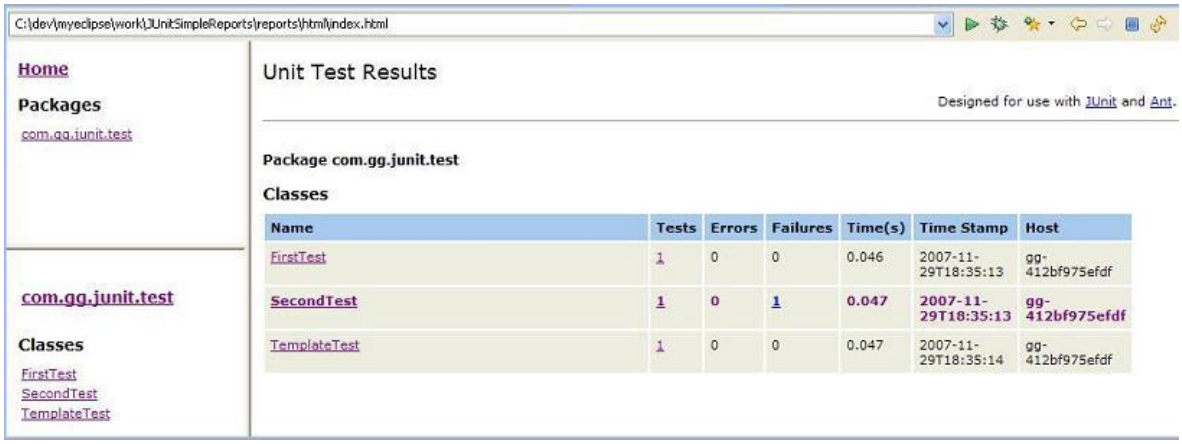


Figure 4 : JUnit Report

2) TestNG Report

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

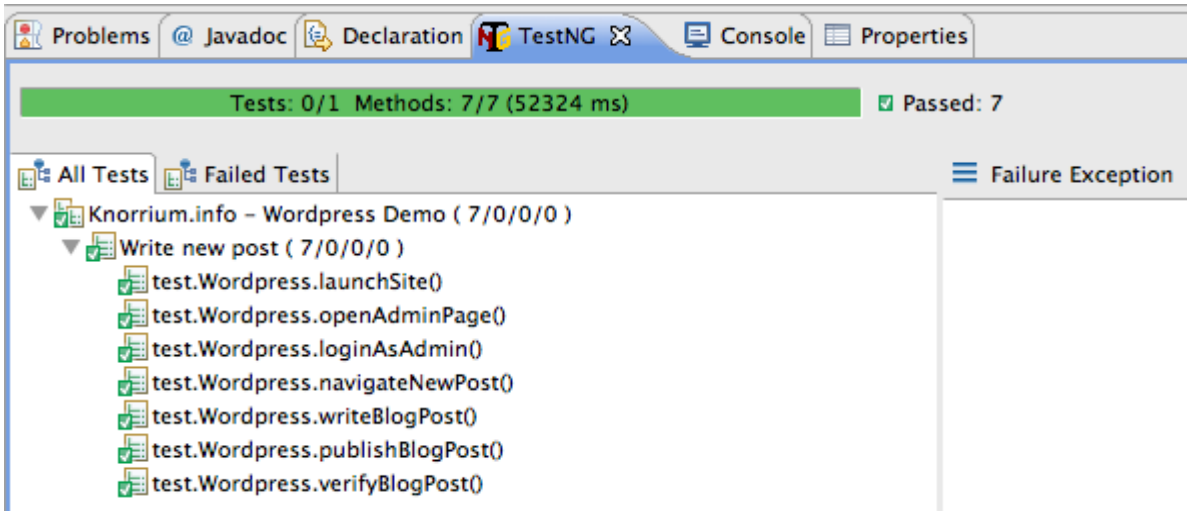


Figure 5 : TestNG Report

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.