

6-2015

Reducing Software Testing Time with Combinatorial Testing and Test Automation

Akalanka Bandara Mailewa
St. Cloud State University

Follow this and additional works at: https://repository.stcloudstate.edu/csit_etds



Part of the [Computer Sciences Commons](#)

Recommended Citation

Mailewa, Akalanka Bandara, "Reducing Software Testing Time with Combinatorial Testing and Test Automation" (2015). *Culminating Projects in Computer Science and Information Technology*. 3.
https://repository.stcloudstate.edu/csit_etds/3

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

Reducing Software Testing Time with Combinatorial Testing and Test Automation

by

Akalanka Bandara Mailewa

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science in

Computer Science

June, 2015

Starred Paper Committee:
Jayantha Herath, Chairperson
Andrew Anda
Susantha Herath

Abstract

The development of large software systems is a complex and error prone process. Errors may occur at any stage of software development. These errors, sometimes referred to as bugs, can cause great losses in terms of both time and money if not identified and removed as early as possible. Testing a software product is costly, since it takes much time and need to test many combinations of its functions, integrity, performance etc., which can be called as test cases. The company's goal is to reduce the testing time, so that they can save money and deliver the product much faster to the customer. Testing time can be reduced in two main ways, first by reducing number of test cases and second by automating repeatedly testing areas. This paper will discuss fundamentals of testing such as importance and difference of verification and validation, testing throughout the software development life cycle and testing methods, levels and types. Then it will discuss the possibility of reducing the time spent on testing by reducing number of test cases with combinatorial testing and automating repeatedly tested areas with test automation using Selenium tool. Finally it will also shed some light on a real world test automation project with selenium and two integrated develop environments.

Table of Contents

	Page
List of Tables	5
List of Figures	6
Chapter	
1. Introduction	8
Software Verification and Validation	9
Problem Statement	9
First Solution	10
Second Solution	11
2. Basics of Software Testing	15
Manual Testing vs. Automated Testing	16
Agile Software Development and Testing	19
Agile QA Process	20
Agile Testing Methodologies	21
Software Testing Life Cycle	23
Testing Methods, Levels and Types	26
Testing Methods	26
Testing Levels	27
Testing Types	29
3. Combinatorial Testing	33
Forms of Combinatorial Testing	36

Chapter	Page
Averages of Combinatorial Testing	40
Disadvantages of Combinatorial Testing	40
When to Use Combinatorial Testing	40
When Not to Use Combinatorial Testing	40
Test Tools Incorporating Combinatorial Testing	41
4. Test Automation with Selenium and JAVA	43
Selenium Tool	43
Selenium IDE	43
Selenium Client API	48
Selenium Remote Control	48
Selenium Grid	50
Selenium WebDriver	50
Sample Test Cases Automation Using Selenium IDE	53
Sample Test Cases Automation Using Selenium WebDriver	56
5. Conclusion	63
References	66
Appendix	69

List of Tables

Table	Page
2.1 The difference between manual testing and automated testing	18
2.2 Comparison between different scripting techniques	19
2.3 Different phases of STLC	25
4.1 Steps of test case execution and expected outcomes	61

List of Figures

Figure	Page
1.1 Pairwise combinatorial test case selection versus manual test case selection (a) Testing efficiency and (b) testing quality	11
1.2 Manual versus automated tests cost comparison	13
2.1 The Agile QA process	20
2.2 Phases of SCRUM	22
2.3 Phases of extreme programming	23
2.4 Software Testing Life Cycle	24
3.1 Cumulative percent of faults vs. the number of parameters	34
3.2 A dialog box with 10 options	36
3.3 A single test case containing many triplets	37
3.4 Arrangement of test cases	37
3.5 Arrangement of a single triplet	38
3.6 Arrangement of two triplets	38
3.7 Arrangement of three triplets	39
4.1 Selenium-IDE	44
4.2 Test case runs against a URL	46
4.3 Test case runs against a second URL	46
4.4 Outline of Selenium RC's architecture	50
4.5 An auto-generated directory structure by Maven	52
4.6 Login with valid username and password	53
4.7 Login with invalid username and valid password	54

Figure	Page
4.8 Login with valid username and invalid password	55
4.9 Open Eclipse IDE	56
4.10 Eclipse IDE with JAVA, Selenium and Maven	57
4.11 Configurations of Apache Maven POM file to install dependencies	58
4.12 When actual outcome is equal to the predicted outcome	59
4.13 When actual outcome is unequal to the predicted outcome	60
4.14 Automation of purchase a book from a web site to illustrate locating elements	62

Chapter 1: Introduction

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product meets the business and technical requirements, works as expected and can be implemented with the same characteristics [1], in other words testing means comparing the actual outcome and expected outcome.

Testing is a never ending process, i.e., it cannot be said that the particular software is 100% error free. Therefore we have to define when to start and when to stop testing. In order to maintain and manage entire testing process it is always better to have a Software Testing Life Cycle (STLC) inside the Software Development Life Cycle (SDLC).

According to ANSI/IEEE 1059 standard, testing is defined as a process of analyzing software item to detect the differences between existing and required conditions and to evaluate the features of the software item [2]. Testing can also be defined as a process of evaluating a system or its component(s) in order to find whether it satisfies the specified requirements. Finally testing comes down to finding the difference between expected outcome and actual outcome. Software testing can also provide an objective, independent view of the software to identify and highlight the risks of software implementation.

The main objective of software testing is to find software bugs, i.e., errors or other defects. The word 'bug' came into use with a real bug, an insect causing a computer program to malfunction. Removing the bug in order to correct the problems was called debugging, a word that is commonly used today but has nothing to do with real insects. Software testing

ensures that; software meets the specified requirements, works as expected, can be implemented as expected, and produces the expected outcome [3].

One of the major advantages of software testing is that it eliminates or reduces errors, leading to faster development time and less cost. Thus theoretically, the final product can be delivered on time with a reasonable price tag. Therefore testing is beneficial for the client as well as the developer.

Software Verification and Validation

Software testing methods can broadly be classified as verification and validation. In a nut shell, while verification answers the question 'Are we building the right thing?' validation focuses on the question 'Are we building the thing right?'. Verification is the process of evaluating products in development phase to determine whether they meet the specified requirements for that phase. Plans, requirement specs, design specs, code, test cases, etc. are used for verification. Reviews, Walkthroughs and Inspections are the methods employed for verification. Validation is the process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements. The actual software is used for validation.

Problem Statement

Effectiveness and Efficiency plays a major role in software testing. Test effectiveness is the relative ability of testing strategy to find bugs in the software. Test efficiency is the relative cost of finding a bug in the software under test. One of the major problems in software industry is testing time. Time means money and hence the entire process is costly. As stated earlier, software testing eliminates or reduces errors leading to faster development

time and lower cost. Unfortunately as mentioned above testing itself can be time consuming and hence costly. Combined with frequent modifications, comprehensive testing can be never ending for large-scale software. Valuable time that could be used for programming has to be allocated for testing. Therefore, to enjoy the benefits of testing, it is important to reduce the time spent on testing without sacrificing the quality of the testing process. There are two major approaches to this end: one is to reduce time spent on testing by reducing the number of test cases, and the other, to automate repetitive test steps thereby increasing testing speed leading to reduced testing times.

First Solution

In order to reduce number of test cases it is used combinatorial testing as a solution to first problem. Combinatorial testing allows you to minimize the number of test cases but yet give a result close to comprehensive testing. The minimal set of test cases is called the covering array. The covering array is selected based on the maximum number of interactions between different inputs that can lead to errors. Interaction of two inputs (or configurations) is often considered when testing which is known as pairwise testing. Figure 1.1 (below) shows summary of results from 10 projects, a comparison between manual and pairwise testing in terms of speed and effectiveness.

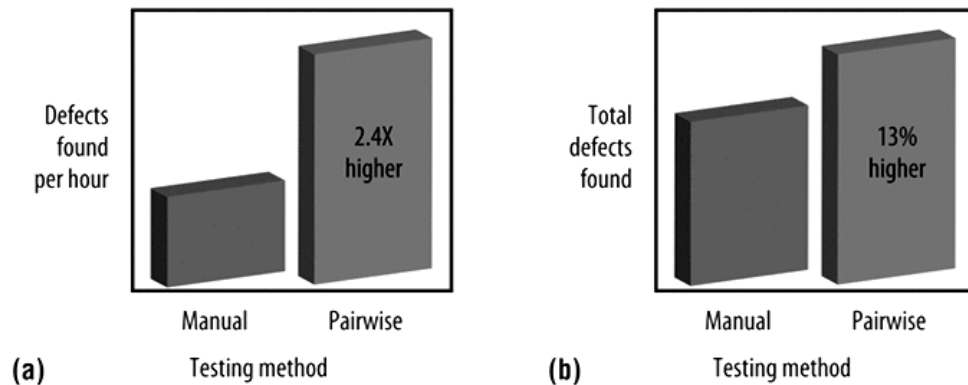


Figure 1.1

Pairwise combinatorial test case selection versus manual test case selection: (a) testing efficiency and (b) testing quality [4]

Although pairwise testing is commonly employed it did not shed much light on the concept of combinatorial testing. Here we use 3-way interactions and show how the covering array reduces hundreds of apparently required test cases to a mere handful enabling us to conduct testing in a fraction of the expected time. Combinatorial testing can detect hard-to-find software faults more efficiently than manual test case selection methods. While the most basic form of combinatorial testing-pairwise-is well established, and adoption by software testing practitioners continues to increase, industry usage of these methods remains patchy at best. However, the additional training required is well worth the effort [4]. By using this method number of test cases can be minimized and hence can be reduced the testing time. This paper discuss combinatorial testing with more detail in Chapter 3 and this is the first way of achieving author's goal.

Second Solution

Another method to cut testing costs is to increase the speed of testing in a different manner and this will give a solution to the second problem. If similar parts of code are to be

tested repetitively, the process can be automated. Test automation plays a very important role in the software testing process due to time, cost and other circumstances, exhaustive testing is not feasible that's why there is a need to automate the testing process to faster the testing of repeatedly testing areas [5]. In Chapter 4 the paper focuses on functional testing of web based applications with browser integrated tool. Finally by automating commonly and mostly testing areas of web based application, it will be able to reduce testing time and so that can be saved testing cost. This is the second way of archiving author's goal. There are several test automation methods and tools available for different automation purposes. This paper will consider about functional test automation of web based applications by using Selenium tool with Java in Eclipse and IntelliJ-IDEA integrated development environments.

Researchers found that tests which are executed a few number of times only during the entire lifetime of the product are usually not worth spending automation resources on. On the other hand, it may be well worth automating the tests that are executed many times as shown in Figure 1.2, for example, tests used for extensive regression testing of areas of high-risk. Of course it is more practical to have a mix of manual and automated tests.

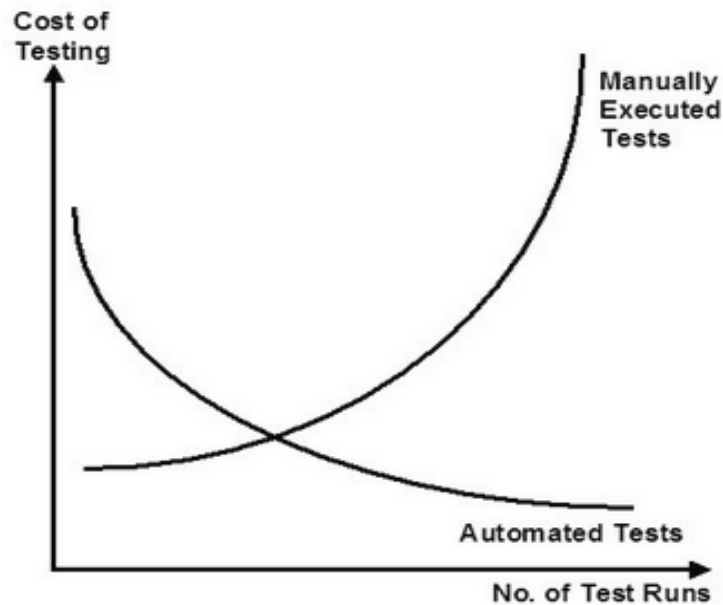


Figure 1.2

Manual versus automated tests cost comparison [6]

The cost associated with selection, implementation, and maintenance of the software testing tool is quite significant. It generally includes expenses incurred on: selection of tool, procurement of tool (use open source, buy or develop internally), licenses, customization, implementation, training of personnel, tool usage, maintenance of automated test ware and tool maintenance. Some of these expenses are measured directly in terms of money; while others come from time spent by the team members [6].

This chapter introduced software testing by discussing the aims and importance of software testing. Then it addressed the two main problems of software testing and suggested two solutions in order to reduce the number of test cases and testing time using combinatorial testing and test automation accordingly. Chapter 2 will discuss basics of software testing. Chapter 3 presents the solution to first problem using combinatorial testing while Chapter 4

providing solution to second problem with the art of the test automation in detail with some examples.

Chapter 2: Basics of Software Testing

It is described in Chapter 1 about brief introduction to the software testing, in this chapter it is explained in detail the process of software testing start by answering three main questions in software testing; 1) who does testing, 2) when to start testing, and 3) when to stop testing. Rest of the chapter describes concepts used in testing.

While large IT companies have a test team, their smaller counterparts may have only a single tester. Some organizations hire software testers from outside or outsource testing process. The testers' responsibility is to evaluate the software with the given requirements. Sometimes developers and testers may conduct unit testing where they may separately test each individual component. Unit testing can take place before the whole system has been developed and is discussed in detail later in this chapter. People involved in testing of a system are software testers, software developers, project lead/manager and end users. Different companies have difference designations for the testers such as software tester, software quality assurance (QA) engineer, and QA analyst, etc. [2]. This is on the basis of their experience and knowledge and this paragraph answered the first question.

As an answer to the second question, it is mentioned that an early start to testing reduces the cost and time of rework. In SDLC, testing can start from the requirements gathering phase and lasts till the deployment of the software. However it also depends on the development model that is being used. For example in waterfall model formal testing is conducted in the testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested [2]. Testing is done in different forms at every phase of SDLC and following situations can also be considered as testing.

- Analysis and verifications of requirements during requirement gathering phase.
- Reviewing the design in the design stage.
- Testing performed by a developer on completion of each section of code.
- Indirect testing by the end user after implementation.

As an answer to the third question, it is difficult to determine when to stop testing as software cannot be tested 100%. Therefore testing is a never-ending process. Even when the end user is working with the software, it is being tested. Following are the aspects which should be considered to stop the testing [2].

- Deadlines set forth for testing.
- Completion of execution of test cases.
- When testing costs are so high that testing is not financially viable.
- Completion of functional and code coverage to a certain point.
- No high priority bugs identified and overall bug rate is below a predefined level.
- Management decision.

The remainder of the chapter will present; first the difference of manual testing vs. automated testing, second about agile software testing, next about the STLC and finally testing methods, levels and types in the STLC.

Manual Testing vs. Automated Testing

Software testing can be divided into two main categories, manual testing, and automated software testing. Both categories have their individual strengths and weaknesses. In this section, paper will present in detail about these two testing techniques and comparison of each other.

With a manual testing, the more traditional approach, tester initiates each test, interacts with system, reports and evaluate the test results. To satisfy the test results manually, testers should prepare and execute test cases on system under test (SUT). These test cases will best test the system using defined processes trying to find bugs. So, they can be fixed before releasing the product to the public [7].

Automation is one of the more popular and available strategies to reduce testing effort. It develops test scripts that will be used later to execute test cases instead of human [8]. The idea behind automation is to let computer simulate what the tester is doing in reality when running test cases manually on SUT. Automated software testing (AST) is more suitable for repetitive tasks during different testing levels such as regression testing, where test cases are executed several times whenever the source code of SUT is modified or updated [9].

Manual testing is more appropriate for finding new and unexpected bugs. Automated software testing is more suitable to prevent new errors in the already tested working modules. Automated testing can perform a large number of test cases in little time, whereas manual testing uses the knowledge of the tester to target testing to the parts of the system that are assumed to be more error-prone. In this sense the two approaches are complementary to each other. [7].

Test scripts are the basic element of automation. Test script is a series of commands or events stored in a script language file to execute a test case and report the results. It may contain logical decisions that affect the execution of the script, creating multiple possible pathways, constant values, variables whose values change during playback. The advantage of test scripts development process is that scripts can repeat the same instruction many times in

loops, each time with different data. There are many types of scripting techniques that can be used in automation such as linear, structured, shared, data-driven and keyword-driven. Table 2.2 shows a comparison between these scripting techniques.

Executing the test cases manually without any tool support is known as manual testing. Taking tool support and executing the test cases by using automation tool(s) is known as automated testing. Table 2.1 (below) shows the brief but comprehensive explanation of difference between manual testing and automated testing with their advantages and disadvantages.

Table 2.1

The difference between manual testing and automated testing

Manual Testing	Automated Testing
Time consuming and tedious: Since test cases are executed by human resources it is slow and tedious.	Fast execution: Automation runs test cases significantly faster than human testers.
Huge investment in human resources: Test cases need to be executed manually so more testers are required in manual testing.	Less investment in human resources: Test cases are executed by using automation tool(s) so lesser numbers of testers are required in automation testing.
Less reliable: Manual testing is less reliable as tests may not be performed with precision each time because of human errors.	More reliable: Automation tests perform precisely same operation each time they are run.
Non-programmable: No programming can be done to write sophisticated tests which fetch hidden information.	Programmable: Testers can program sophisticated tests to bring out hidden information.
The results are late: programmers cannot see the result until the tester note down, type, print and copy the results.	Quick results: the programmers can see the result real-time in a networked environment.
Might catch more of the errors occurring due to human nature as the testers are human beings themselves.	Might skip errors occurring due to human nature as software cannot 'think' as human beings.
Can be cheaper for small software than automated testing.	Can be costly for smaller projects but cost-effective for larger projects.
Much better at visual testing.	Weak at visual testing. Software doesn't think like humans so it cannot decide a GUI is attractive, distractive or dull.

Table 2.2

Comparison between different scripting techniques [10]

Property	Linear	Structured	Shared	Data-Driven	Keyword-Driven
Ability to use reusable functions	No	No	Yes	Yes	Yes
Data separation from test script	No	No	No	Yes	Yes
Logic steps separation from test script	No	No	No	No	Yes
Access to code required	No	Yes	Yes	Yes	Yes
Use structured programming instructions	No	Yes	Yes	Yes	Yes
Ability to compare test results with expected	No	Yes	Yes	Yes	Yes
Ability to using script in regression testing	No	Yes	Yes	Yes	Yes
Special framework required	No	No	No	No	Yes
Programming skills level	1 (Low)	2	3	4	5 (High)
Effort needed to create test script	1 (Low)	2	3	4	5 (High)
Maintenance costs needed to update test script	5 (High)	4	3	2	1 (Low)
Reusability of test script	1 (Low)	2	3	4	5 (High)

Agile Software Development and Testing

The word agile more commonly suggests a focused yet rapidly iterative software process adhering to principles that were first outlined in the Agile Manifesto first laid down in 2001. The manifesto aimed to promote a more efficient way of developing computer programs and IT systems through a collaborative system of various teams. Today, the agile method has become widely accepted as an effective approach to project management within the mainstream software development and testing community [11]. Agile testing refers to the practice of testing software for bugs or performance issues within the context of an agile workflow [11], [12].

Agile QA Process

Agile testing comes from the same philosophy as agile methodology and stresses on testing practices which emphasizes individuals and actions over processes and tools. Software products become more valuable than the documentation. Clients become included in the process of creating and testing software [13]. Figure 2.1 shows the process of agile software quality assurance.

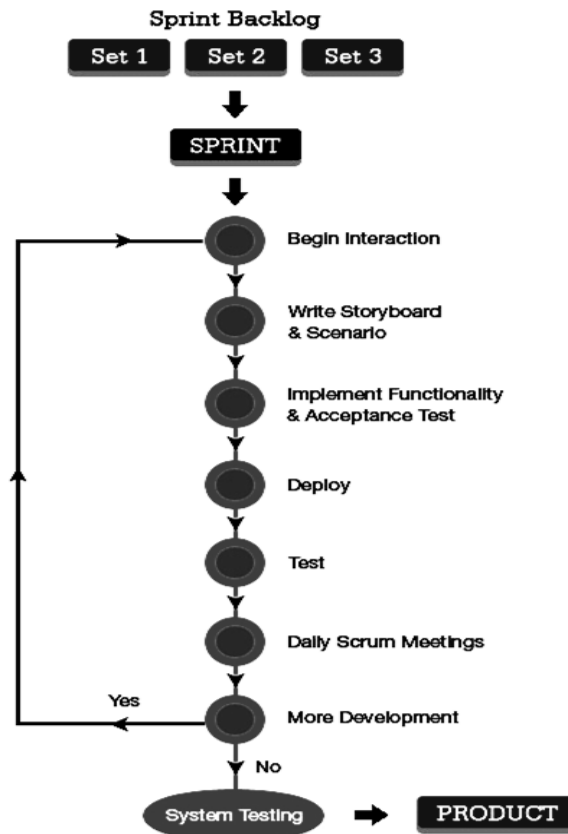


Figure 2.1

The Agile QA process [13]

In the Agile methods, the work cycle becomes abbreviated. Instead of phases and schedules, it has cadence points which result in completed product increments. Less

documentation is required with a focus on code and whether the product piece works. Specification becomes by example with tests for software products based on real examples—a product piece in hand functioning. Every two weeks, the direction of a project receives a review making it easy to steer in other directions as conditions change. Quality becomes part of development and becomes everyone's responsibility. By incrementing, inspecting and adapting development cost and time to market becomes reduced. Continual planning of the release optimizes value and maximizes team ability to compete in the marketplace. Included in the process are business people who have a sense of what users need and want. Change becomes welcomed, constant necessary cooperation fosters communication between differing disciplines and adaptability becomes the norm [12], [13].

Agile Testing Methodologies

Agile Testing has flavors. The method for a particular route of software development must be chosen carefully. Two flavors frequently used by companies are SCRUM, and Extreme programming. A SCRUM framework emphasizes holistic, flexible strategies where development teams work as units on a common goal. Planning and managing a project is decided by a project's operation properties and certainties [13]. Product owner is responsible for the delivery and functionality of three core components, in each iteration development team manages and organizes work in a cycle and Scrum Master who removes all obstacles to progress, sets up Sprint meetings, and sets up the teams. Figure 2.2 shows the Phases of SCRUM.

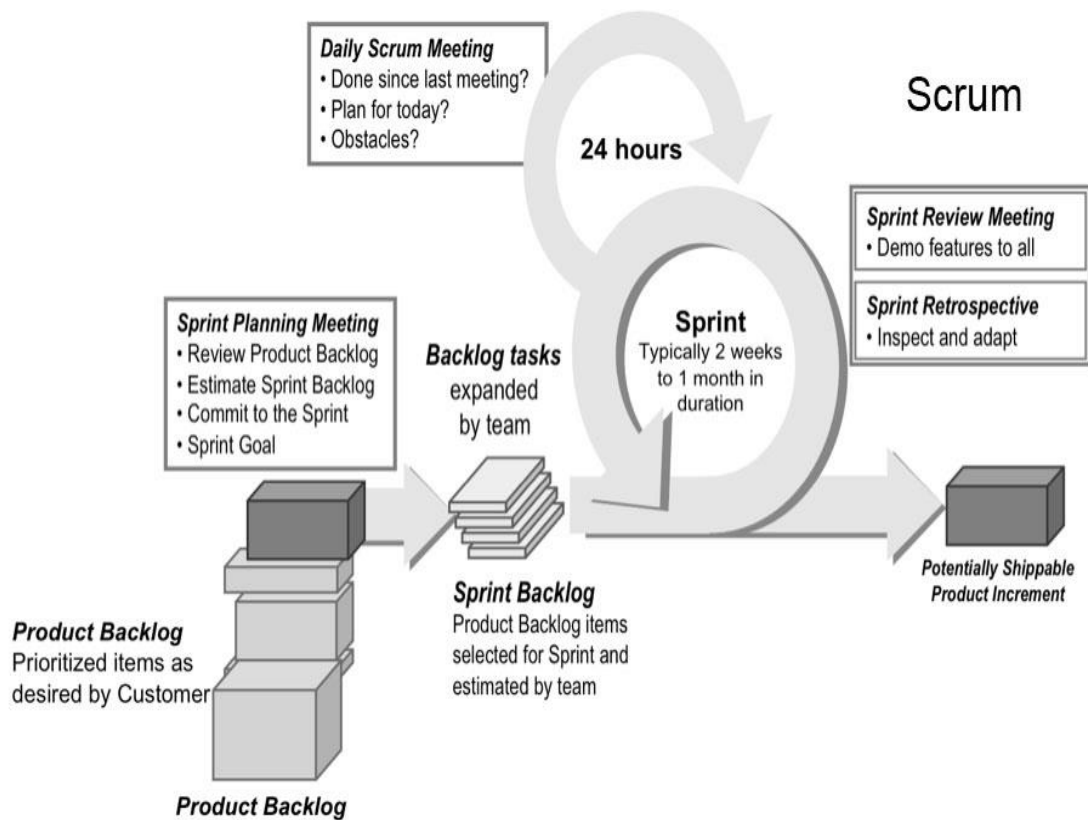


Figure 2.2

Phases of SCRUM [13]

Extreme programming advocate short development cycles on products that focus on improving the product itself or increasing productivity. It works well in environments where clients have constant change [12], [13]. Figure 2.3 shows the phases of extreme programming.

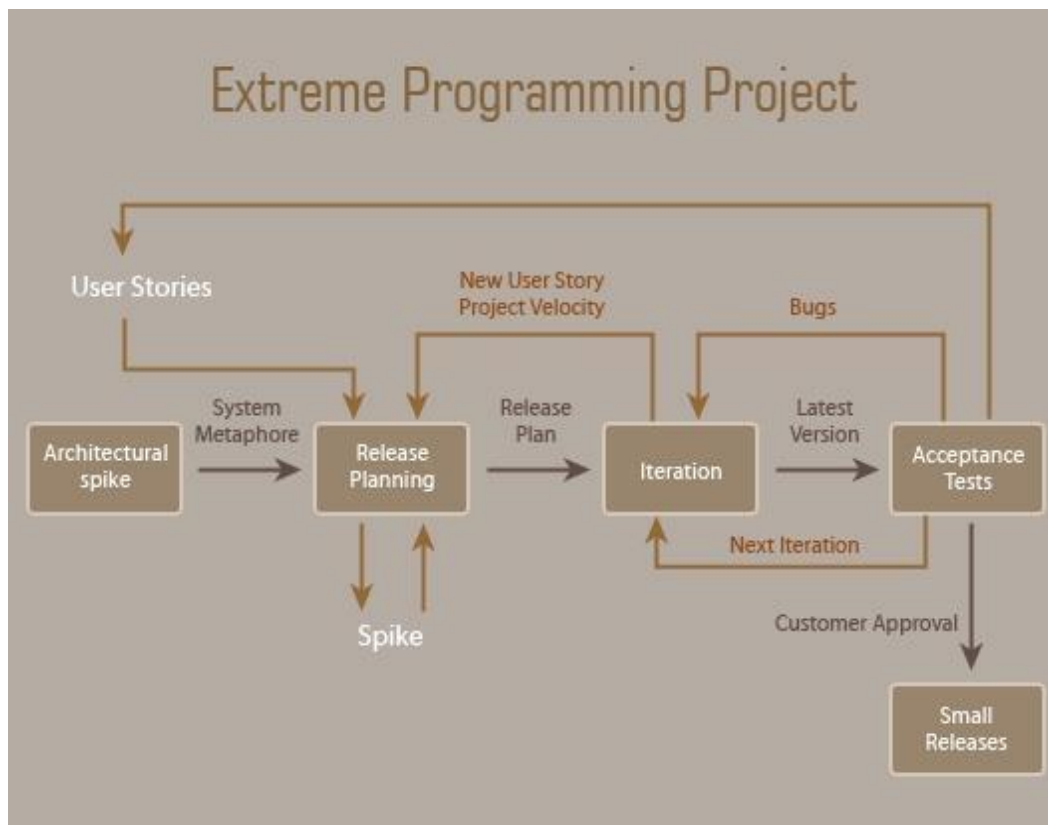


Figure 2.3

Phases of extreme programming [13]

Software Testing Life Cycle

Successful software testing projects depends on a standard testing process, good testing objectives and the right team roles, skills and tools. The automation test team needs to have knowledge of testing, programming, and automation tool as well. In today's system, rapid deployments and quick responses to the customers are essential. Software development and test processes during a product life cycle are adapted to these requirements [14]. To maintain all mentioned above, software testing has its own life cycle by which it follows and checks the software at various stages of SDLC and is known as software testing life cycle (STLC). It is decided by management based on the requirements and development model and

hence there is no any specific universal STLC which applicable to all the development projects. Figure 2.4 show the one of such a STLC diagram and table 2.1 describes all the phases with activities and deliverables.

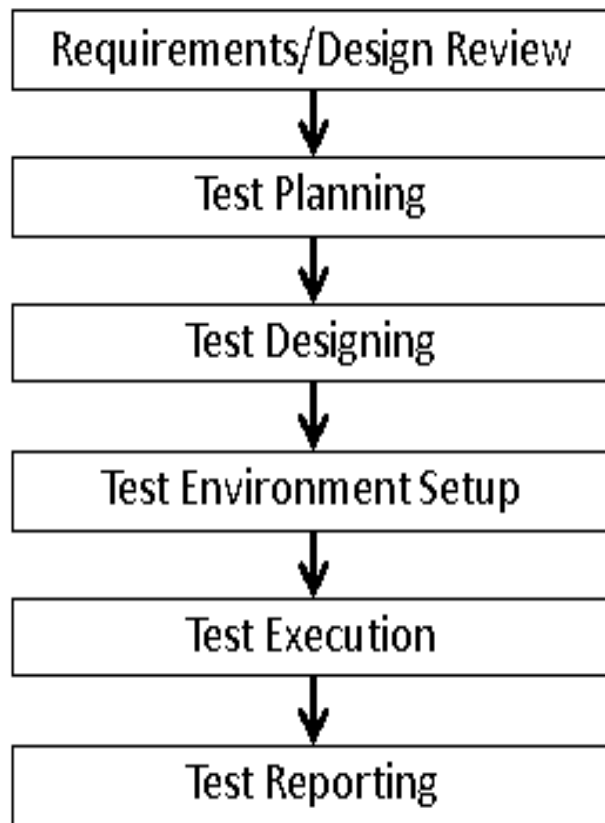


Figure 2.4

Software Testing Life Cycle [15]

Table 2.3

Different phases of STLC [15]

Phase	Activity	Deliverables	Necessity
Requirements/ Design Review	You review the software requirements/ design (Well, if they exist.)	<ul style="list-style-type: none"> Review Defect Reports 	Curiosity
Test Planning	Once you have gathered a general idea of what needs to be tested, you 'plan' for the tests.	<ul style="list-style-type: none"> Test Plan Test Estimation Test Schedule 	Farsightedness
Test Designing	You design/ detail your tests on the basis of detailed requirements/design of the software (sometimes, on the basis of your imagination).	<ul style="list-style-type: none"> Test Cases / Test Scripts /Test Data Requirements Traceability Matrix 	Creativity
Test Environment Setup	You setup the test environment (server/ client/ network, etc.) with the goal of replicating the end-users' environment.	<ul style="list-style-type: none"> Test Environment 	Rich company
Test Execution	You execute your Test Cases/ Scripts in the Test Environment to see whether they pass.	<ul style="list-style-type: none"> Test Results (Incremental) Defect Reports 	Patience
Test Reporting	You prepare various reports for various stakeholders.	<ul style="list-style-type: none"> Test Results (Final) Test/ Defect Metrics Test Closure Report Who Worked Late & over time 	Diplomacy

The STLC phases mentioned above do not necessarily have to be in the order listed; some phases can sometimes run in parallel (For instance, Test Designing and Test Execution). And, in extreme cases, the phases might also be reversed (For instance, when there is Cursing prior to Testing). Interestingly, no matter how well-defined a Software Testing Life Cycle you have in your project or organization, there are chances that you will invariably witness the widely-popular cycle called Testing and Cursing. In this type of STLC, you skip phases like design review, test planning, etc. In the hope that the skipping will save you some time and/or cost. But, it never does in the real world [15].

Software testing can be implemented at any stage of the Software Development Life Cycle. Testing after the coding has been completed was the traditional method, but in the agile approaches, testing is an ongoing process. There are several test methodologies and which one to choose will depend on the software development methodology chosen [16].

Testing Methods, Levels and Types

In this section the paper will discuss about different testing methods, levels and types which are applicable throughout the software testing life cycle. First discuss briefly about five standard testing methods, second presents seven generally recognized levels of tests with some examples and finally talk about general testing types in two main categories, one is functional testing types and the other is non-functional testing types as a high level overview.

Testing Methods

There are several approaches / techniques of Software Testing. Although there are many software testing methods currently available, this paper will only include the following five methods for the simplicity and to give the basic idea.

Static testing. Involves verification and it usually asks or checks "Are you building the thing right?" It is primarily syntax checking of the code or manually reviewing the code, requirements documents, design documents etc. to find errors [17], [18].

Dynamic testing. Involves validation and it usually asks or checks "Are you building the right thing?" The software should be compiled and executed and input values are given and output values are checked with the expected output [17], [18].

Black-box testing. Treats the software as a "black box" with inputs and outputs [18]. The tester is only aware of what the software is supposed to do, not how it does it. Therefore it is also known as Specification-based testing technique or input/output driven testing techniques.

White-box testing or 'glass-box' testing. Internal structures or workings of a program are tested. It is also known as Structure-based technique because here the testers require knowledge of how the software is implemented.

GUI testing. This is the process of testing a product's graphical user interface to ensure it meets its written specifications. This type of testing is common when designing web pages where sizes and alignments of images and buttons are checked [19].

Testing Levels

Each phase of SDLC goes through the testing hence there are various levels of testing. Although there are many levels of software testing currently available, this paper will only include the following seven main levels for the simplicity and to give the basic idea.

Unit testing. This is a method by which individual units of source code are tested together with associated control data. A unit is the smallest testable part of an application.

Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended [20].

Component testing. Individual software modules are tested in order to find defects and to verify their proper functionality. Component testing is also known as module and program testing. Component testing may be done in isolation from rest of the system depending on development life cycle model chosen for that particular application [21].

Integration testing. Individual software modules are combined and tested as a group to verify they work together without errors. Integration testing is done after unit testing and prior to validation testing. It is done by a specific integration tester or a test team [21].

System testing. Conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and no knowledge of the inner design of the code or logic is required. System testing investigates both functional and non-functional requirements of the software [22].

Acceptance testing. Once all or most of the defects have been corrected after the system test, the system will be delivered to the user or customer for acceptance testing. It is conducted to determine if the specified requirements are met prior to its delivery. Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well [20].

Alpha testing. Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often used for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing. It takes place at the developer's site [23].

Beta testing or field testing. Takes place at customer's site. The software is installed and tested under real-world working conditions. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team [23]. This is very common in game development.

Testing Types

A test type is focused on a particular test objective. Although there are many types of software testing currently available, according to the scope and organization of this paper, the paper will discuss only the following types. Testing types are mainly divided into two categories as functional and non-functional testing.

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work." Some functional testing types are described below [18], [20]. This paper explains only Installation testing, Development Testing, Destructive testing, Recovery testing, User acceptance testing types in detail but there are some other types of functional tests are available which are not discussed in detail such as; Usability, Sanity, Smoke, Regression and Automated.

Installation testing. Assures that the system is installed correctly and working at actual customer's hardware.

Development testing. Software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer

or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Development testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Destructive testing. Attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines [2]. Software fault injection, in the form of fuzzing, is an example of failure testing. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform destructive testing.

Recovery testing. Ability of software to recover from crashes and errors due to hardware problems, timeouts or unavailability of resources.

User acceptance testing. Performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

Non-functional testing is tests below refer to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution. Non-functional

requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users [18], [20].

Compatibility testing. Conducted to evaluate the application's compatibility with the computing environment. It ensures there will be no conflicts with various hardware of software once the product is implemented. This may include compatibility with hardware, operating systems, other linked application software, network system, web browser(s), linked database(s) etc.

Performance testing. Generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Security testing. This is essential for software that processes confidential data to prevent system intrusion by hackers.

Accessibility testing. Include testing the compliance with standards such as Americans with Disabilities Act of 1990, Section 508 Amendment to the Rehabilitation Act of 1973 and/or Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C).

Internationalization and localization. General ability of software to be internationalized and localized can be automatically tested without actual translation, by using pseudo-localization. It will verify that the application still works, even after it has been translated into a new language or adapted for a new culture (such as different currencies or time zones) [24].

This chapter presented the basics of software testing with some important points such as when to start and when to stop testing in the industry standard projects, showed the difference between manual and automated testing, gave an introduction to agile software methodologies and testing. Finally this chapter discussed importance of software testing in software development life cycle, the basics of software testing and different stages of STLC by introducing various methods, levels and types used in software testing. Chapter 3 will discuss combinatorial testing in detail and Chapter 4 will discuss test automation with Selenium and Java.

Chapter 3: Combinatorial Testing

Combinatorial testing is a relatively new form of software testing. It is based on simple principles such as interaction rule, pairwise testing, t-way interaction testing, etc. Combinatorial testing can drastically reduce testing times and costs by reducing the number of tests (i.e., reducing number of test cases) to be performed.

Sometimes even reliable software might fail when certain unusual combination of inputs are entered. This could have been avoided if all the inputs had been tested previously. But this is a tedious task when the number of possible inputs is high. This kind of exhaustive testing is no longer viable in many situations as modern day software may require hundreds of inputs.

Empirical research shows that majority of bugs occur due to interaction of a few inputs. Combinatorial testing allows us to select two or three parameters at a time, be it configurations or inputs, and test the combinations in such a way that majority of the faults will come to light with minimum number of tests [25].

Interaction rule. Most software failures occur due to faults of a single parameter or interaction of two. Progressively fewer faults occur by interaction of higher number of parameters [25], as can be seen below.

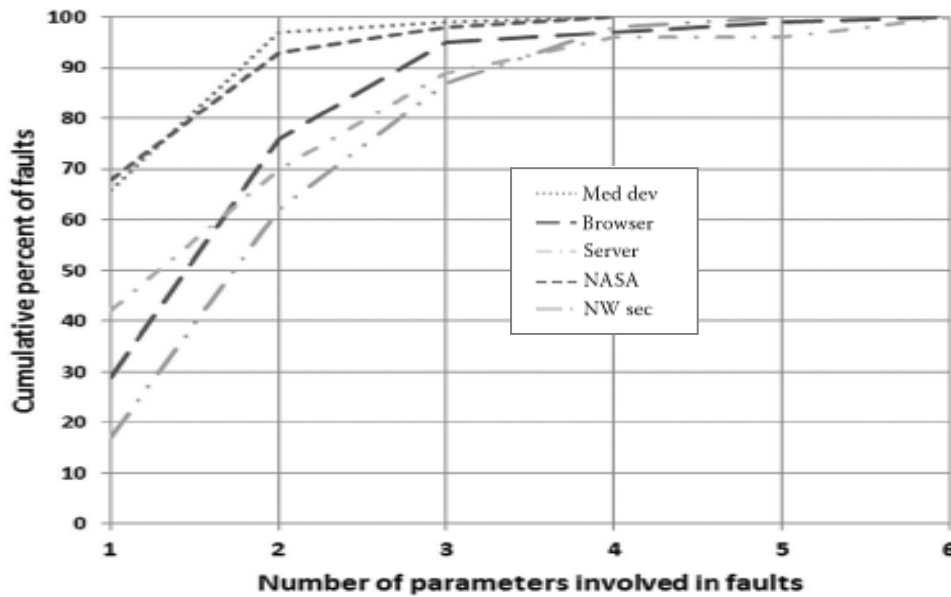


Figure 3.1

Cumulative percent of faults vs. the number of parameters [26]

Figure 3.1 shows empirical research data of cumulative percent of faults vs. the number of parameters involved in faults. The five graphs represent data on medical devices, web browser tests, HTTP server tests, NASA database and Network security tests. The topmost graph is based on 15 years of data gathered on FDA medical device recall data by NIST. When the data from medical devices are considered, it can be seen that approximately 66% of faults occur due to a single factor while around 97% of faults are resulted from single factors or combination of two factors [25].

Pairwise testing. Experimental data shows that about 60-95% of faults occur due to interaction of two parameters. So if all 2-way combinations are tested, a high percentage of errors could be detected. Testing 2-way combinations of configurations and/or inputs is known as pairwise testing [26].

Three-way interactions and higher. However pairwise testing might not be sufficient depending on the application of the software. For example, nobody would want 3% of the patients attached to medical devices to die just because the software was tested only using pairwise testing. In such critical cases, interaction between a higher numbers of factors should be considered. Empirical research shows consideration of a maximum of 6 factors (i.e., 6-way interactions) is enough to identify 100% of faults in almost all cases. This is evident in the above figure too. In certain cases, considering 4-way interactions of input/configuration parameters is sufficient to cover all faults. For non-critical software, testing of 3-way interactions would suffice. The number of interactions taken into account is called the strength of the test.

Difficulties related with higher number of interactions. Pairwise testing is fairly easy and there are also test tools that generate inputs for such testing. Many testers are familiar with pairwise testing. But tests with higher number of interactions, known as higher strength tests, are difficult to perform. The number of combinations to be tested can be fairly high, and there were no reliable testing tools up until recently.

Methodology used in combinatorial testing. In combinatorial testing, a minimum number test for t-way interactions, where “t” is the number of parameters that might generate faults with interacting with each other, is calculated. This number of tests should cover all possibilities of t-way parameter interactions [27].

Forms of Combinatorial Testing

In combinatorial testing, both configurations and/or inputs can be taken as parameters. The two methods are known as configuration testing, and input parameter testing see the combinatorial testing example given bellow to have a better idea how does it work.

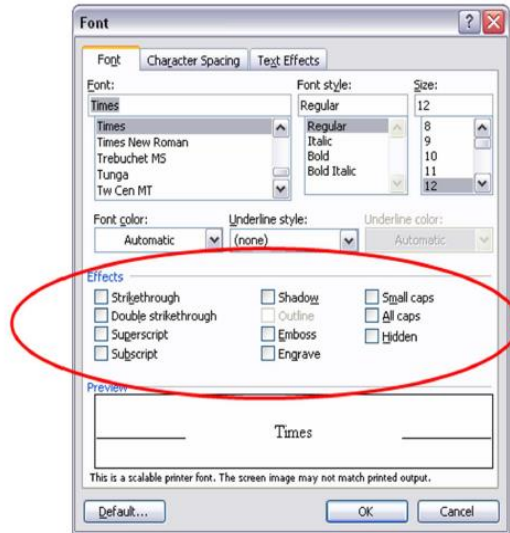


Figure 3.2

A dialog box with 10 options [25]

Figure 3.2 shows a font formatting dialog of word processing software. There are 10 (many of them mutually independent) effects that can be switched on or off. Therefore there are 2^{10} (=1024) combinations. Testing all these combinations is costly. As discussed earlier, the strategy employed in combinatorial testing is to reduce the number of tests to be performed. Assuming 3-way interactions will reveal all the faults, let us decide the number of tests.

- Number of tests = $\binom{10}{3} = 120$ tests.
- Each triplet (three parameters selected for a test) can be on/off producing 2^3 possibilities. That is $120 \times 2^3 = 960$ tests.

- Since three triplets can be used in one test (when 10 inputs are selected), the maximum number of tests is 320.
- But in practice, many triplets can be packed into one test [28].

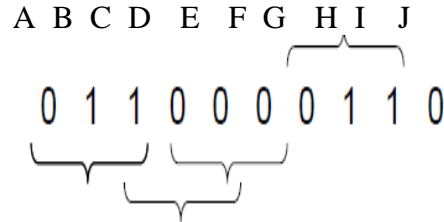


Figure 3.3

A single test case containing many triplets [25]

Consider Figure 3.3. The test case apparently contains three triplets—ABC, DEF and GHI. But actually there are many more triplets contained in this set of inputs. For example BCD, EFG and HIJ can be considered as another three triplets. So are ABD, ABE and ABF. Let us look at this phenomenon in more detail with a sample test array consisting of 13 test cases.

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Figure 3.4

Arrangement of test cases [25]

Let us find the minimum number of tests required for the arrangement of test cases given in Figure 3.4.

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Figure 3.5

Arrangement of a single triplet [25]

Consider Figure 3.5. The first three columns (A, B, C) show three effects. All of the eight combinations possible with these three inputs are present in the above table.

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Figure 3.6

Arrangement of two triplets [25]

Now consider Figure 3.6. Here, three columns D, E, G have been selected randomly (light gray circles). The eight combinations possible with those three inputs are also included.

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Figure 3.7

Arrangement of three triplets [25]

Consider Figure 3.7. Here each column represents a parameter and each row a test case. When observed closely we can see there is not only three, but many triplets are there. In fact, all possible combinations of the 10 parameters are included in these 13 rows. Hence we can deduce that the minimum number of tests required is only 13. Compare this with the number of exhaustive tests, 1024, or presumed number of 120 tests [27].

Covering array. Designing the proper array of test cases is the tricky part in combinatorial testing. By properly arranging the test cases we can find the minimum number of tests that cover all t-way combinations (3-way, in the above example). This arrangement of parameters and test cases is known as covering array. Automated testing tools that employ combinatorial testing can quickly create the covering array [28].

Advantages of Combinatorial Testing

Combinatorial testing offers several advantages. First of all it decreases the testing costs by reducing the number of tests needed. It allows faster fault detection as the number of test cases is minimal. By both means it saves considerable amount of time. With combinatorial testing rare conditions can be tested, that might be omitted in conventional testing methods. Combinatorial testing is highly efficient with higher strength tests. Recent developments in automated testing tools provide solutions for problems that need testing the interaction of 4 or more variables which would have been next to impossible several years ago.

Disadvantages of Combinatorial Testing

Combinatorial testing can be costly at higher strength interactions (>4-way) when done manually. It can be time consuming even when automated (especially with very high strength tests). Manual form of combinatorial testing requires high skill levels, e.g., for developing the covering array.

When to Use Combinatorial Testing

Combinatorial testing is the best choice when there are many parameters (inputs/configurations) are present and errors occur due to interactions between these parameters. It could be employed when the system concerned is a critical system but exhaustive testing is impossible due to certain circumstances.

When Not to Use Combinatorial Testing

Combinatorial testing will not be useful when there are very few number of parameters where exhaustive testing is possible. It is also useless when there are no

interactions between parameters. Similarly combinatorial testing should not be employed when interactions among different inputs/configurations do not lead to additional errors.

Test Tools Incorporating Combinatorial Testing

A combinatorial test design (CTD) algorithm finds a small test plan that covers 100% of a given interaction level [27]. Five tools have been introduced by National Institute of Standards and Technology (NIST) for combinatorial testing.

1. Advanced Combinatorial Testing System (ACTS)—generates test sets that ensure t-way coverage of input parameter values; includes support for constraints and variable-strength tests.
2. Combinatorial coverage measurement—computes a number of coverage measures of an existing test set.
3. Access Control Policy Test (ACPT)—uses combinatorial testing with model checking to produce tests for access control policies.
4. .NET Configuration test file generator—provide combinatorial coverage for .NET systems that have a large number of configuration options.
5. Combinatorial sequence test generator—generates sequence covering arrays, useful for event driven systems including GUIs, protocols, hardware testing.

In addition, IBM has developed a software named Functional Coverage Unified Solution (FoCus) for combinatorial testing purposes.

This chapter discussed combinatorial testing and software tools that use combinatorial testing. How combinatorial testing reduces thousands of test cases to a few is interesting and

it might be the future of software testing. Chapter 4 will be on test automation with Selenium and Java by using some other software tools.

Chapter 4: Test Automation with Selenium and JAVA

In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes [29]. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually. This chapter will discuss in detail the functional and interface (GUI) automation using the selenium IDE and WebDriver tools with JUNIT, Java, Eclipse IDE and FireBug.

Selenium Tool

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. It is open-source software, released under the Apache 2.0 license, and can be downloaded and used free of charge [30].

Selenium IDE

Selenium IDE (Figure 4.1) is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On, and allows recording, editing, and debugging tests. It was previously known as Selenium Recorder [31].

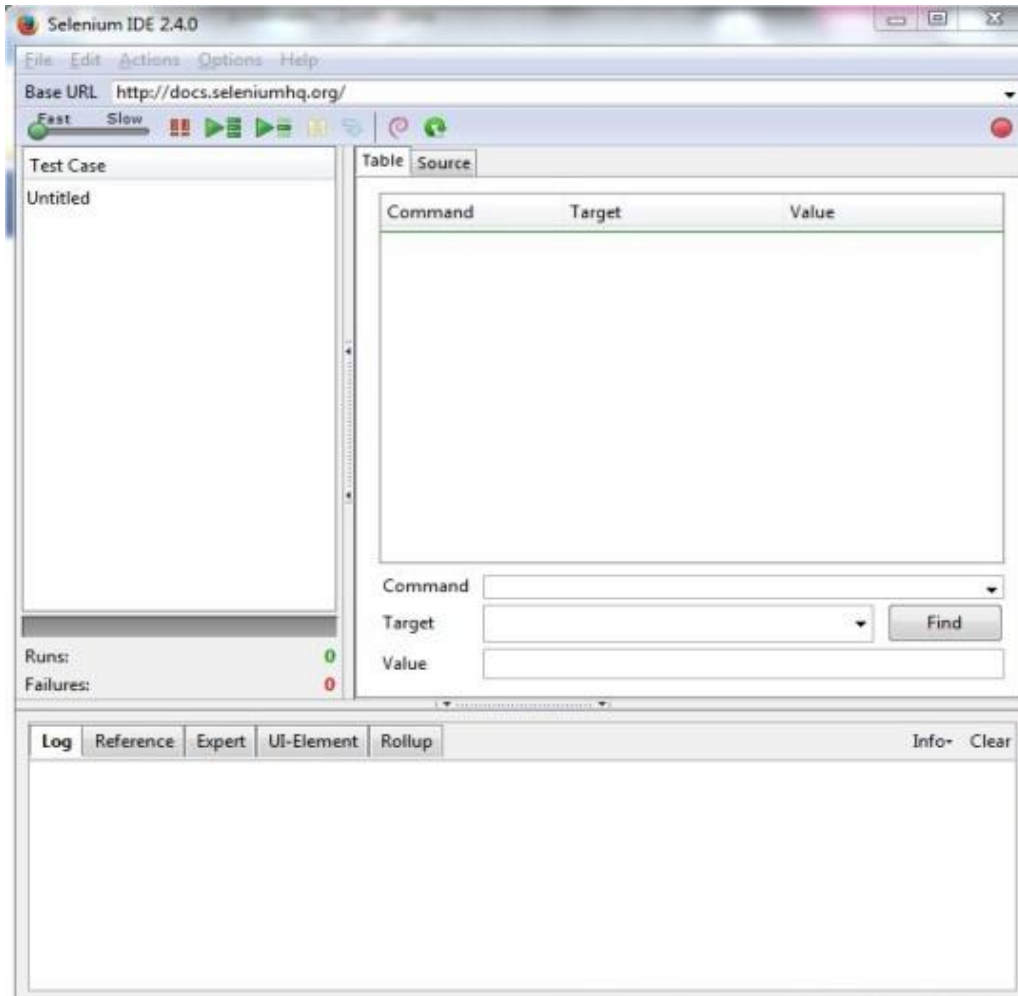


Figure 4.1

Selenium-IDE

Recording. Many first-time users begin by recording a test case from their interactions with a website. When Selenium-IDE is first opened, the record button is ON by default. If you do not want Selenium-IDE to begin recording automatically, turn it off under Options > Options... and deselecting “Start recording immediately on open” [32]. During recording, Selenium-IDE will automatically insert commands into your test case based on your actions. Typically, this will include:

- clicking a link - click or clickAndWait commands
- entering values - type command
- selecting options from a drop-down listbox - select command
- clicking checkboxes or radio buttons - click command

Here are some tricky situations to be aware of:

- The type command may require clicking on some other area of the web page for it to record.
- Following a link usually records a click command. You will often need to change this to clickAndWait to ensure your test case pauses until the new page is completely loaded. Otherwise, your test case will continue running commands before the page has loaded all its UI elements. This will cause unexpected test case failures.

Using base URL to run test cases in different domains. The Base URL field at the top of the Selenium-IDE window is very useful for allowing test cases to be run across different domains. Suppose that a site named `http://news.portal.com` has an in-house beta site named `http://beta.news.portal.com`. Any test cases for these sites that begin with an open statement should specify a relative URL as the argument to open rather than an absolute URL (e.g., starting with `http:` or `https:`). Selenium-IDE will then create an absolute URL by appending the open command's argument onto the end of the value of Base URL [32]. For example, the test case below would be run against `http://news.portal.com/about.html` (Figure 4.2):

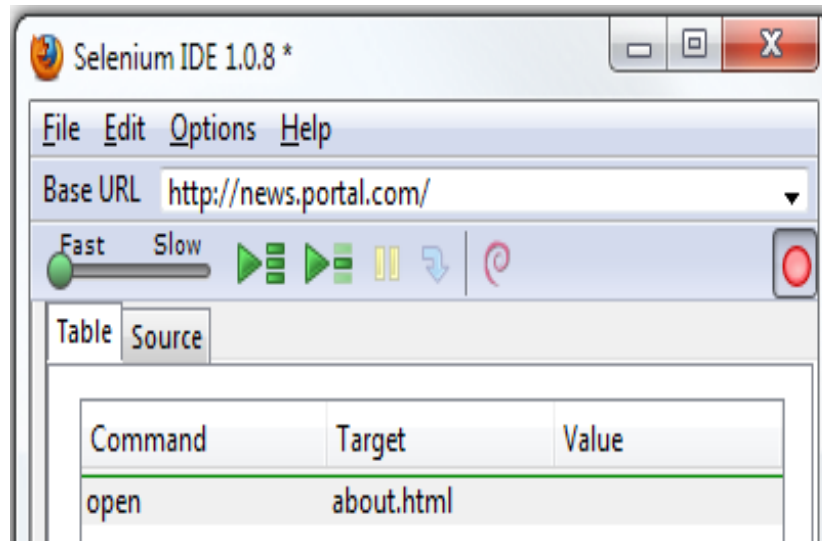


Figure 4.2

Test case runs against a URL

This same test case with a modified Base URL setting would be run against `http://beta.news.portal.com/about.html` (Figure 4.3):

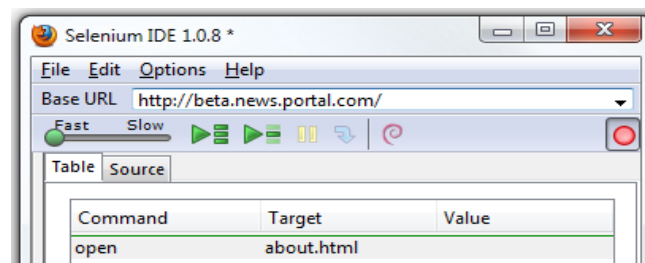


Figure 4.3

Test case runs against a second URL

For many Selenium commands, a target is required. This target identifies an element in the content of the web application, and consists of the location strategy followed by the location in the format “locatorType=location”. The locator type can be omitted in many cases. The various locator types are explained below with examples for each [32].

- **Locating by Id.** This type of locator is more limited than the identifier locator type, but also more explicit. Use this when you know an element's id attribute.
- **Locating by Name.** The name locator type will locate the first element with a matching name attribute. If multiple elements have the same value for a name attribute, then you can use filters to further refine your location strategy. The default filter type is value (matching the value attribute).
- **Locating by XPath.** XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications. XPath extends beyond (as well as supporting) the simple methods of locating by id or name attributes, and opens up all sorts of new possibilities such as locating the third checkbox on the page.

One of the main reasons for using XPath is when you don't have a suitable id or name attribute for the element you wish to locate. You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name. Absolute XPaths contain the location of all elements from the root (html) and as a result are likely to fail with only the slightest adjustment to the application. By finding a nearby element with an id or name attribute (ideally a parent element) you can locate your target element based on the relationship. This is much less likely to change and can make your tests more robust.

- **Locating Hyperlinks by Link Text.** This is a simple method of locating a hyperlink in your web page by using the text of the link. If two links with the same text are present, then the first match will be used.
- **Locating by CSS.** CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents. CSS uses Selectors for binding style properties to elements in the document. These Selectors can be used by Selenium as another locating strategy.

Selenium Client API

As an alternative to writing tests in Selenese, tests can also be written in various programming languages. These tests then communicate with Selenium by calling methods in the Selenium Client API. Selenium currently provides client APIs for Java, C#, Ruby and Python. With Selenium 2, a new Client API was introduced (with WebDriver as its central component). However, the old API (using class Selenium) is still supported [32].

Selenium Remote Control

Selenium Remote Control (RC) is a server that accepts commands for the browser via HTTP. It is written in Java. RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks. To make writing tests easier, Selenium project currently provides client drivers for PHP, Python, Ruby, .NET, Perl and Java. The Java driver can also be used with JavaScript (via the Rhino engine). A new instance of selenium RC server is needed to launch html test case—which means that the port should be different for each

parallel run. However, for Java/PHP test case only one Selenium RC instance needs to be running continuously [33]. Selenium RC components are shown below.

- The Selenium Server which launches and kills browsers, interprets and runs the Selenese commands passed from the test program, and acts as an HTTP proxy, intercepting and verifying HTTP messages passed between the browser and the AUT.
- Client libraries which provide the interface between each programming language and the Selenium RC Server.

Figure 4.4 shows the client libraries communicate with the Server passing each Selenium command for execution. Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands. The browser, using its JavaScript interpreter, executes the Selenium command. This runs the Selenese action or verification you specified in your test script [34].

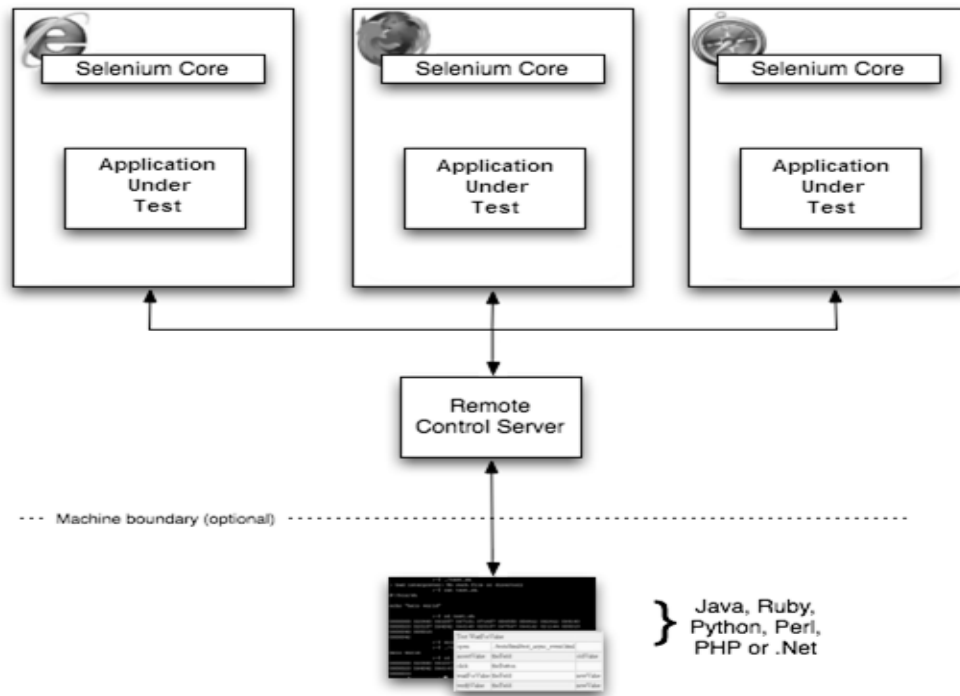


Figure 4.4

Outline of Selenium RC's architecture [30]

Selenium Grid

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (WebDriver nodes), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test) [35].

Selenium WebDriver

Selenium WebDriver is the successor to Selenium RC. Selenium WebDriver accepts commands (sent in Selenese, or via a Client API) and sends them to a browser. This is

implemented through a browser-specific browser driver, which sends commands to a browser, and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox or Internet Explorer); there is also an HtmlUnit browser driver, which simulates a browser using HtmlUnit [35].

Unlike in Selenium 1, where the Selenium server was necessary to run tests, Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. However, Selenium Grid can be used with WebDriver to execute tests on remote systems.

Locating elements can be done as it is described with Selenium IDE such as Locating by Id, Locating by Name, Locating by XPath, Locating Hyperlinks by Link Text and Locating by CSS. In order to find XPath and CSS location it is used FirePath tool in FireBug software which comes as add-on to Mozilla Firefox web browser.

Junit. The unit testing tool Junit mainly deals with fundamental class testing and compounding class testing. However, it cannot conveniently test private methods and domains, neither cannot support testing the method, whose type of parameters is a class, in a fundamental class or a compounding class. In addition, it even cannot efficiently support some special Java class testing, such as the inherited class, abstract class and interface. This paper discusses how to introduce new testing principles and a new plug-in to Junit, and how to extend the source code of Junit to improve the testing effectiveness [36].

FireBug. Firebug is a web development tool that facilitates the debugging, editing, and monitoring of any website's CSS, HTML, DOM, XHR, and JavaScript; it also provides

other web development tools. In this chapter it is used to identify object elements using FirePath tool. It helps in providing the selected object property information [37].

Apache Maven. Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle [38].

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups [38]. Figure 4.5 shows an auto-generated directory structure by Maven.

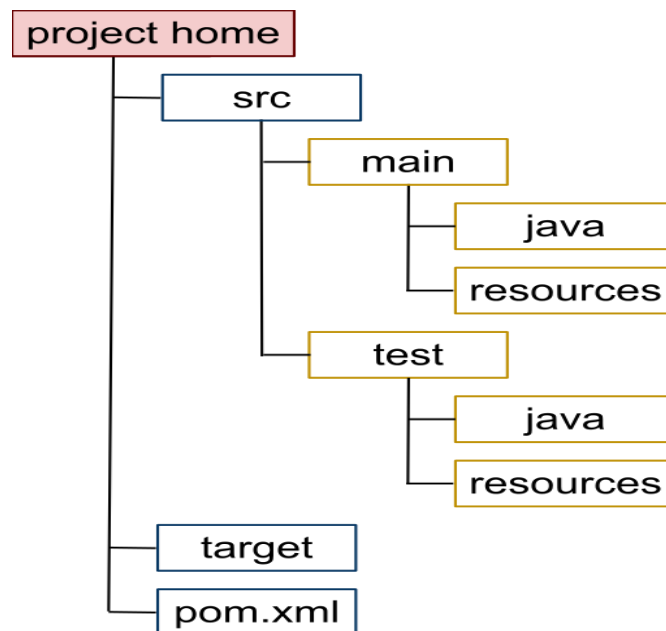


Figure 4.5

An auto-generated directory structure by Maven [38]

Sample Test Cases Automation Using Selenium IDE

TestCase 1. Login with a valid username and a valid password (Figure 4.6).

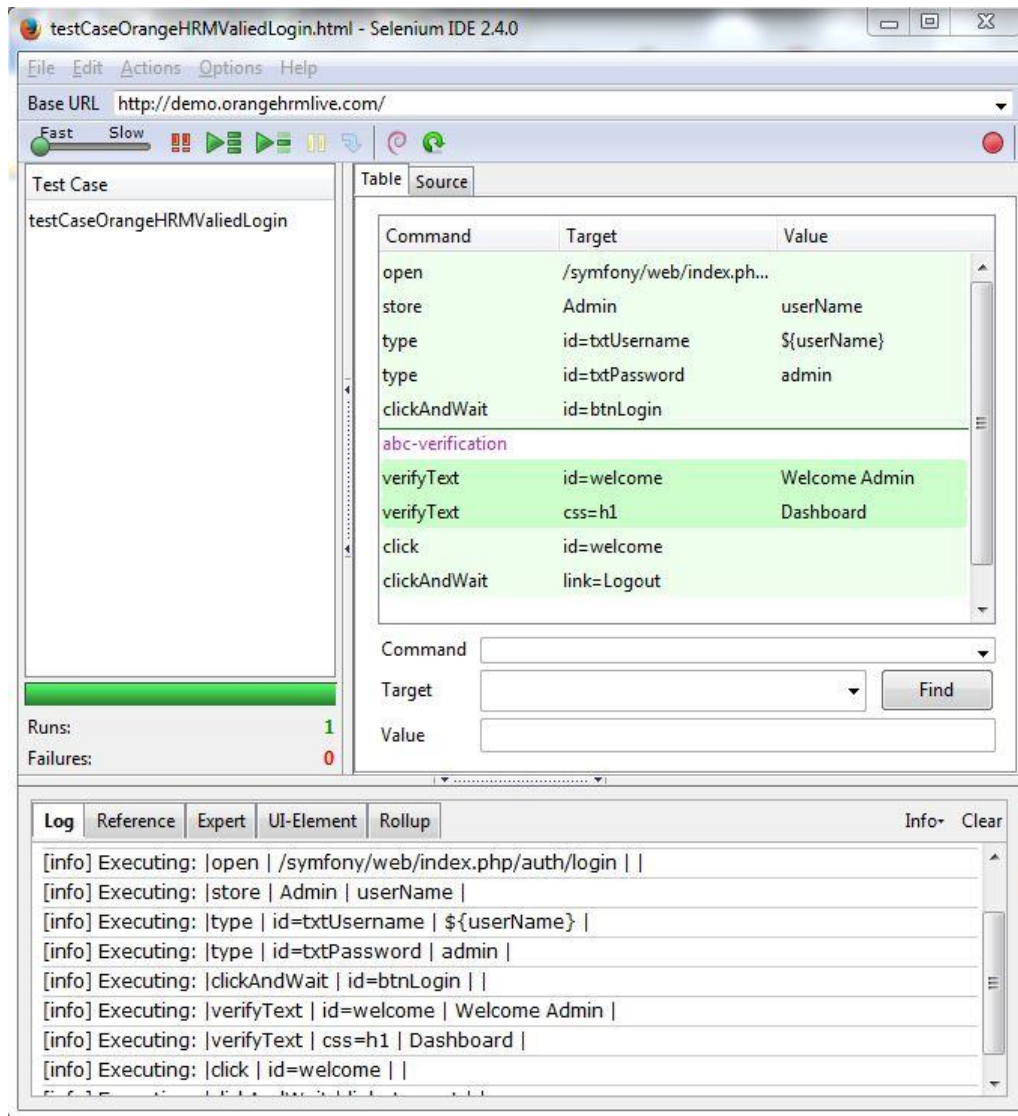


Figure 4.6

Login with valid username and password

TestCase 2. Login with an invalid username and a valid password (Figure 4.7)

The screenshot displays the Selenium IDE 2.4.0 interface. The top bar shows the file name 'testCaseOrangeHRMInvalidUsername.html'. The Base URL is 'http://demo.orangehrmlive.com/'. The Test Case list on the left includes 'testCaseOrangeHRMInvalidUserna...'. The main table shows the following steps:

Command	Target	Value
open	/symfony/web/index.ph...	
store	abc123	userName
click	css=span.form-hint	
type	id=txtUsername	\${userName}
type	id=txtPassword	admin
clickAndWait	id=btnLogin	
verifyText	css=h1	Retry Login
assertTitle	OrangeHRM	

Below the table, there are input fields for Command, Target, and Value, along with a Find button. The bottom section shows the Log of the test case execution:

Log	Reference	Expert	UI-Element	Rollup
[info] Executing: open /symfony/web/index.php/securityAuthentication/				
[info] Executing: store abc123 userName				
[info] Executing: click css=span.form-hint				
[info] Executing: type id=txtUsername \${userName}				
[info] Executing: type id=txtPassword admin				
[info] Executing: clickAndWait id=btnLogin				
[info] Executing: verifyText css=h1 Retry Login				
[info] Executing: assertTitle OrangeHRM				

Figure 4.7

Login with invalid username and valid password

TestCase 3. Login with a valid username and an invalid password (figure 4.8)

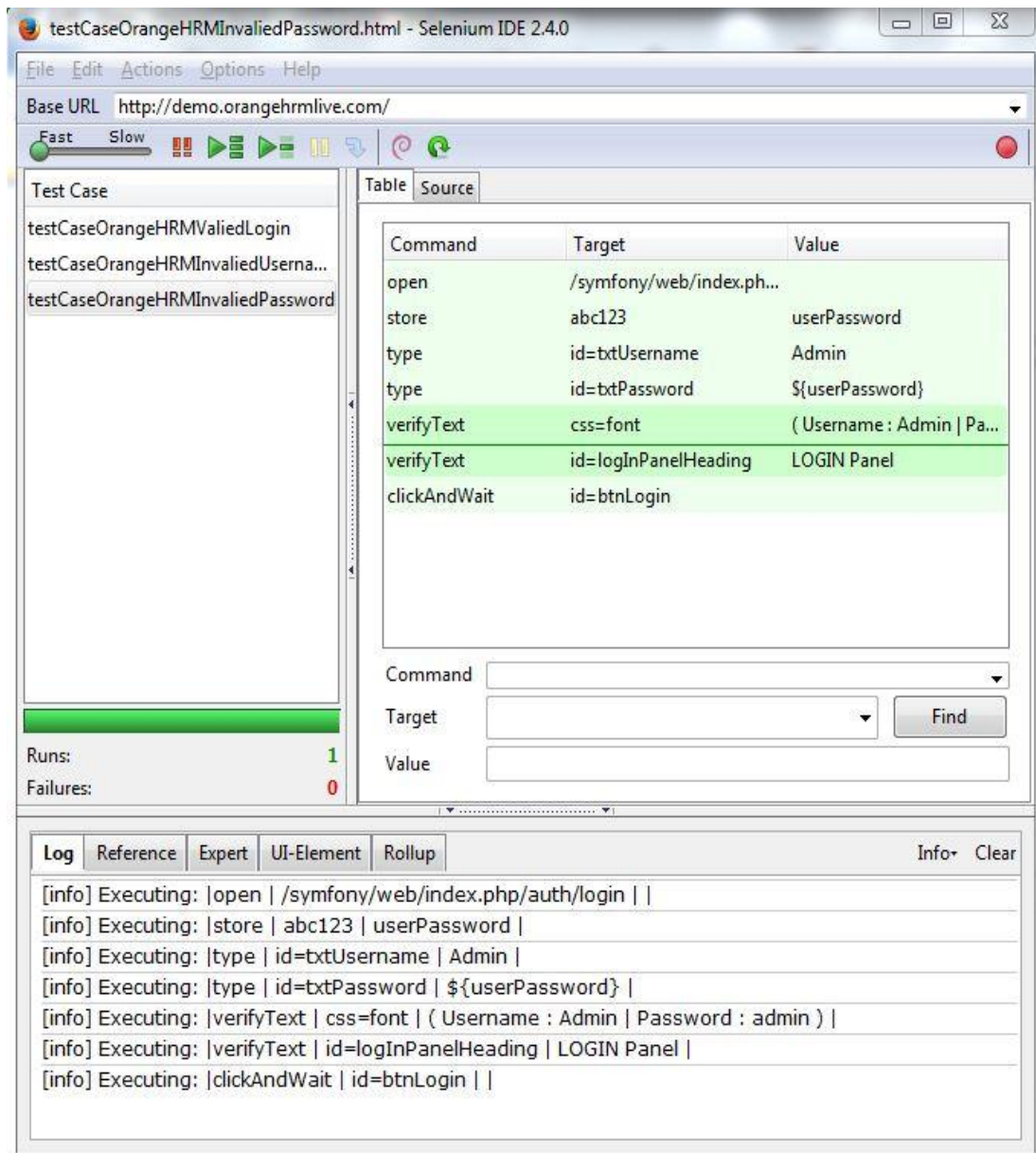


Figure 4.8

Login with valid username and invalid password

Sample Test Cases Automation Using Selenium WebDriver

In order to automate a test case with Selenium WebDriver it is used some additional software such as Java, FireBug, Maven and Eclipse IDE.

Step 1: Open Eclipse IDE When Eclipse IDE is opened, need to set the workspace path (Figure 4.9).



Figure 4.9

Open Eclipse IDE

Step 2: Setup Eclipse environment to use Java, JUNIT, Selenium and Maven
(Figure 4.10).

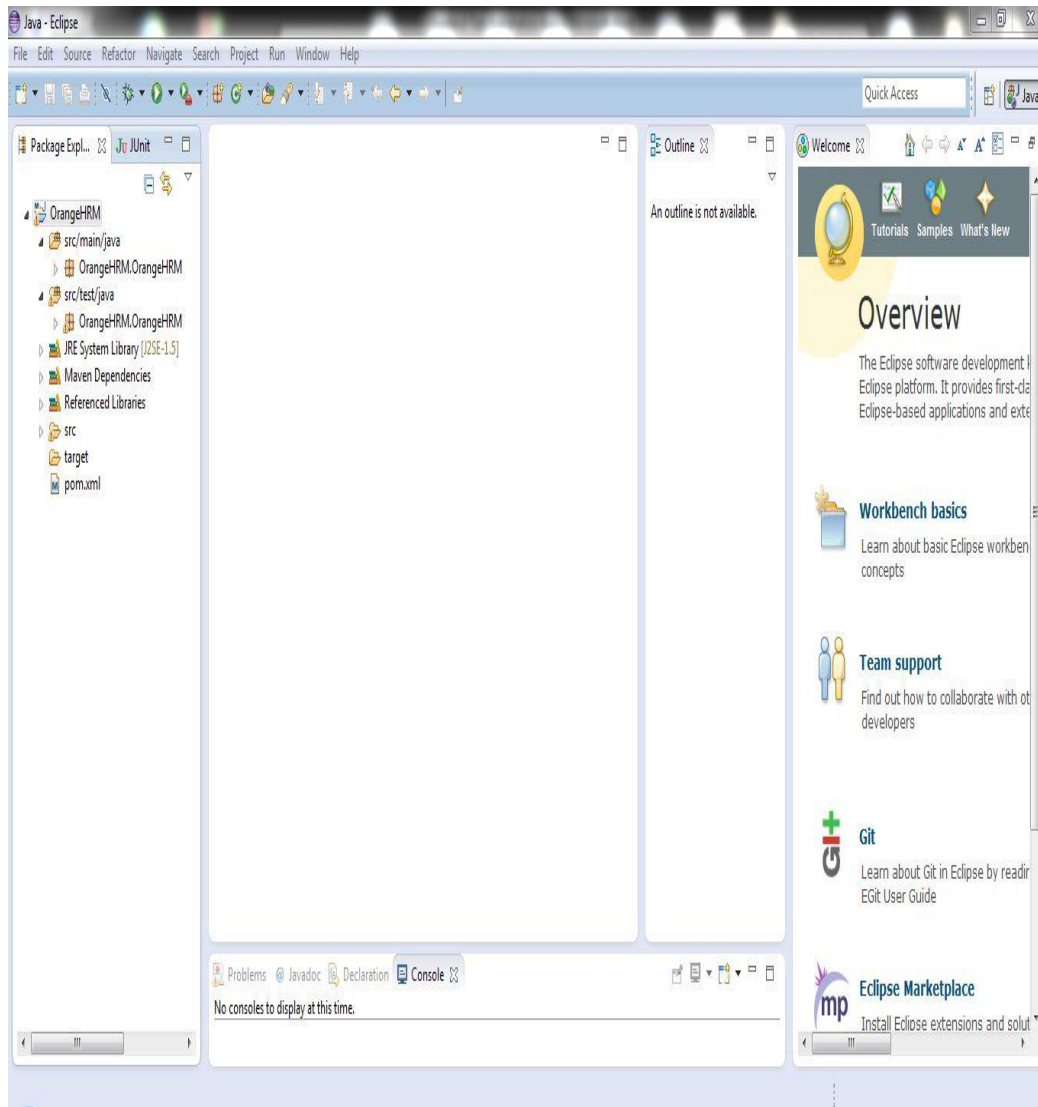


Figure 4.10

Eclipse IDE with JAVA, Selenium and Maven

Step 3: After configuring POM file the framework is ready to automate test cases (Figure 4.11).

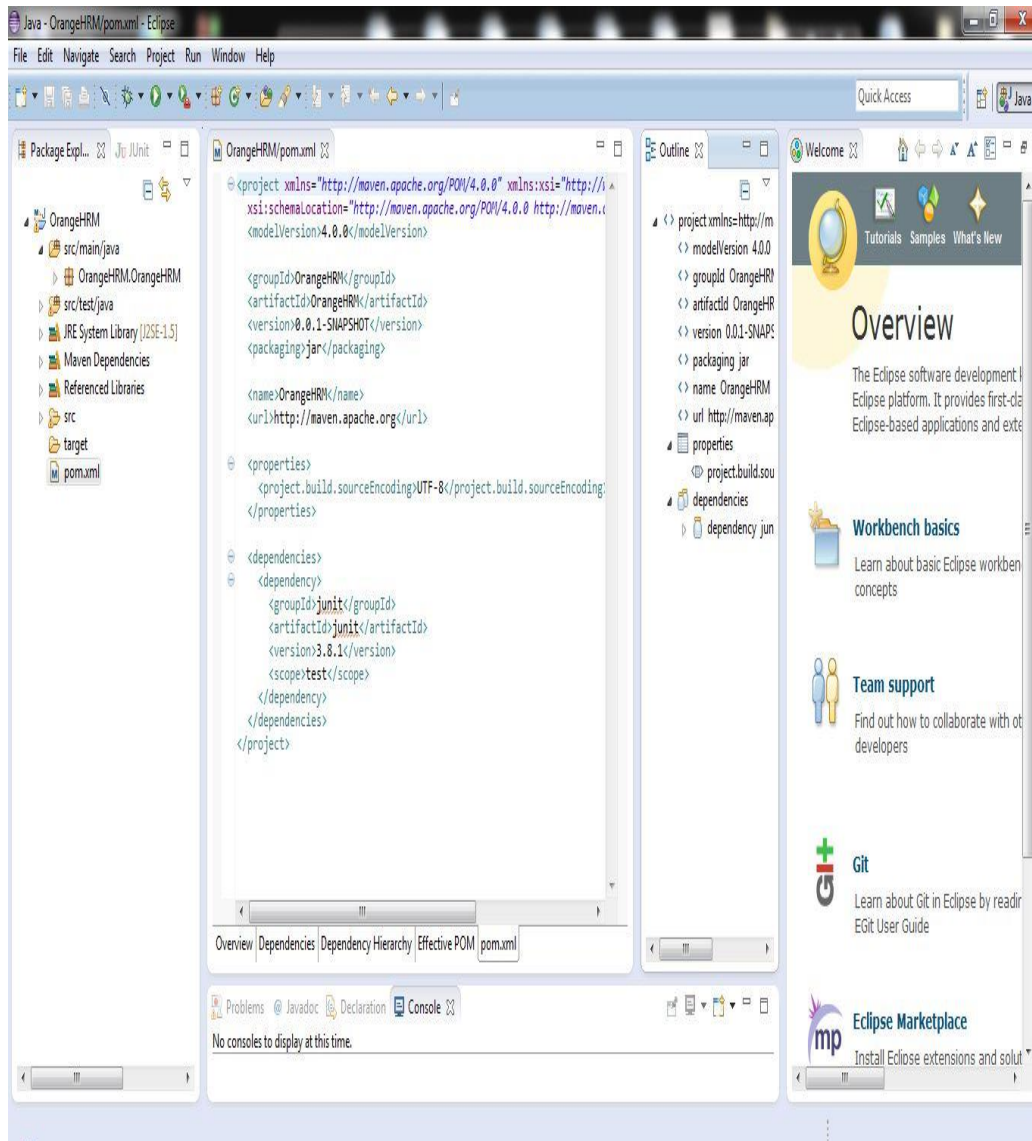


Figure 4.11

Configurations of Apache Maven POM file to install dependencies

Step 4: Sample automated test cases

Test case 1. Go to www.yahoo.com on Mozilla Firefox web browser and check whether the title of the page is 'Yahoo'. If the response is yes (Figure 4.12) then the test is passed. Otherwise the test is failed.

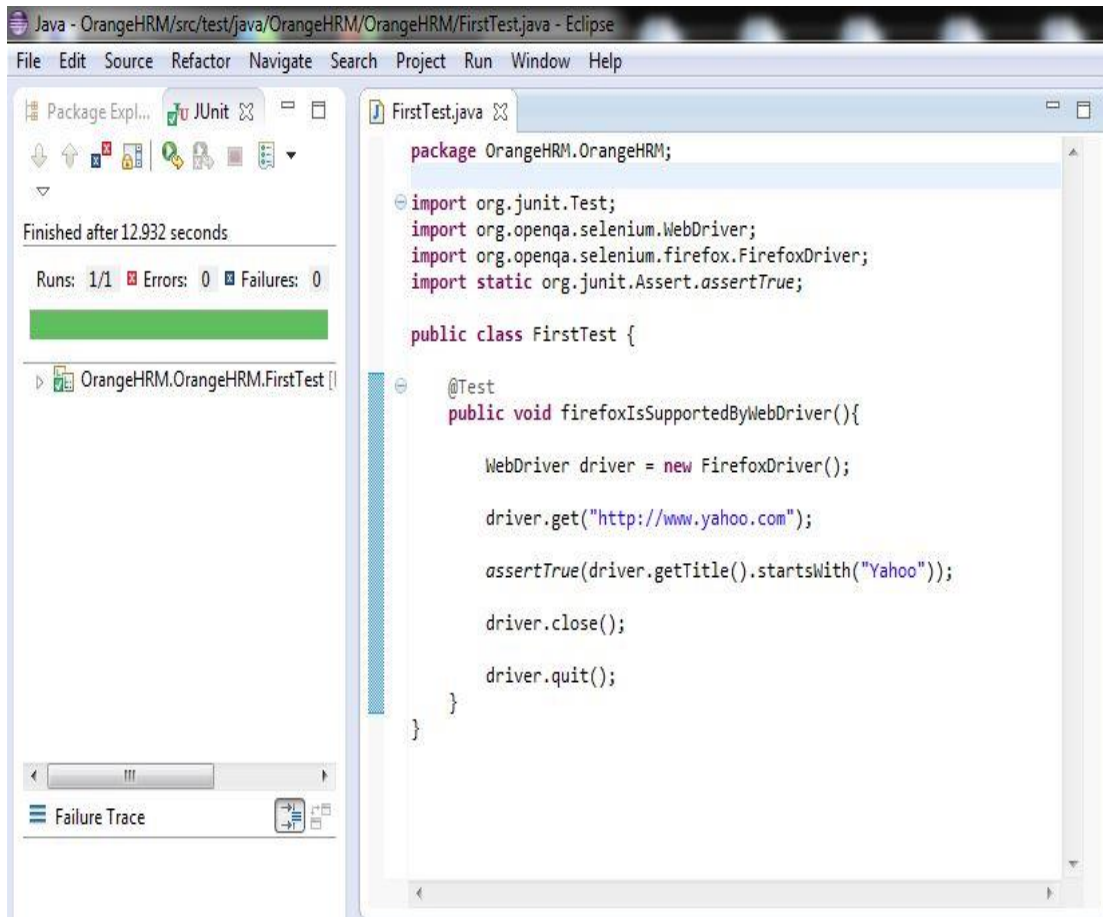


Figure 4.12

When actual outcome is equal to the predicted outcome

Test case 2. Go to www.yahoo.com on Mozilla Firefox web browser and check whether the title of the page is 'Google'. If it is yes test is passed. Otherwise test is failed (Figure 4.13).

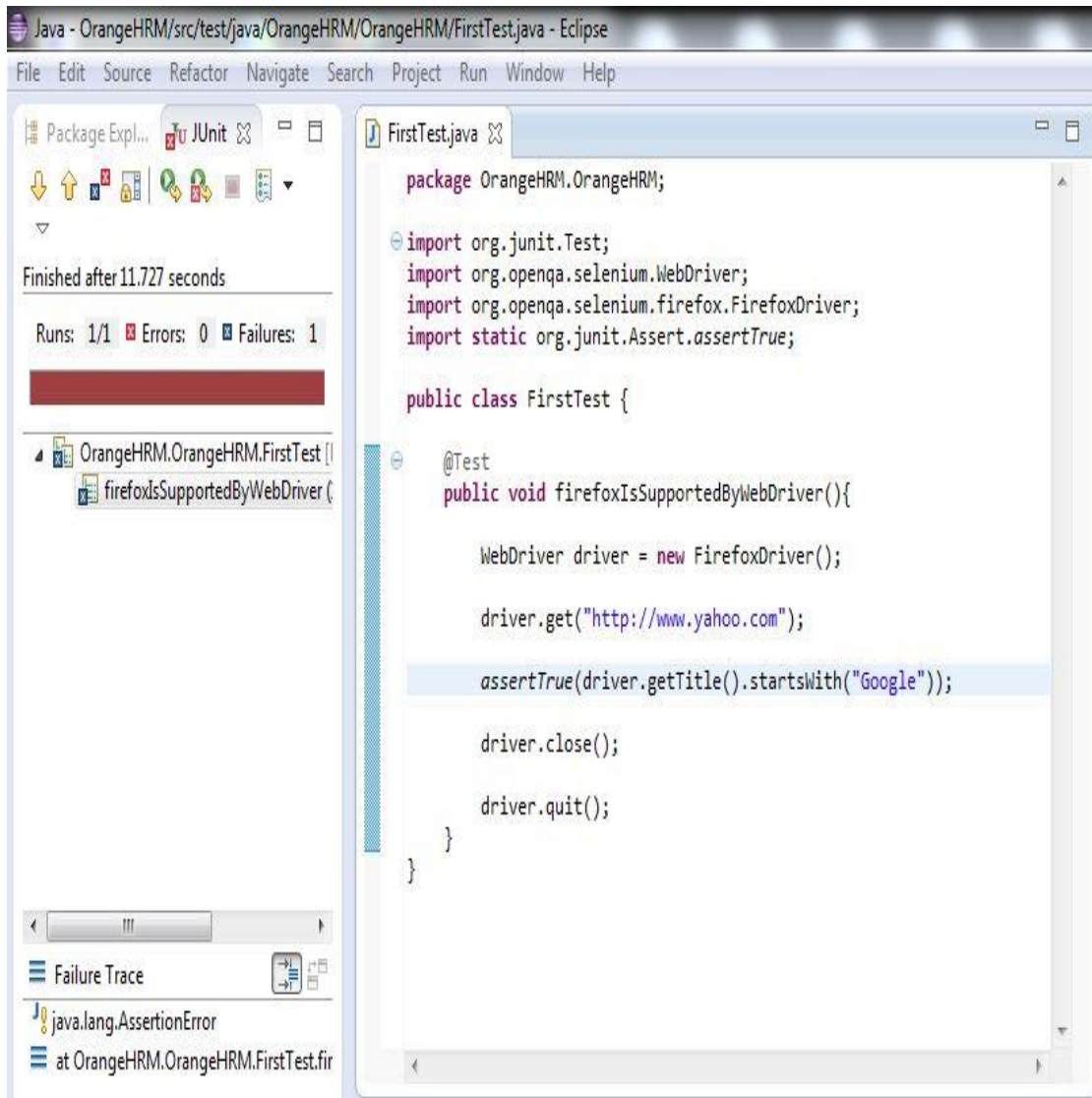


Figure 4.13

When actual outcome is unequal to the predicted outcome

Test case 3. Automate the scenario of purchasing a book from particular online store (http://www.soastastore.com/?page_id=4).

Table 4.1

Steps of test case execution and expected outcomes

Steps:	Expected Outcome
1. Navigate to Soastore.com	User directed to Soastore.com. Page display store.
2. Verify book name 'Vegas Moon'	The book name 'Vegas Moon' is displayed
3. Click Add To Cart	Product Vegas Moon added to cart
4 Verify Total Price is \$14.99	Total Price is \$14.99

In order to automate this test case it is required to find 'Vegas Moon', 'Add to Cart' and 'Total Price' objects on the web page. This example is uses three different locating elements to add above three objects (Figure 4.14).

- Locating Hyperlinks by Link Text to find 'Vegas Moon'
- Locating by XPath to find 'Add to Cart' button click
- Locating by CSS to find 'Total Price'

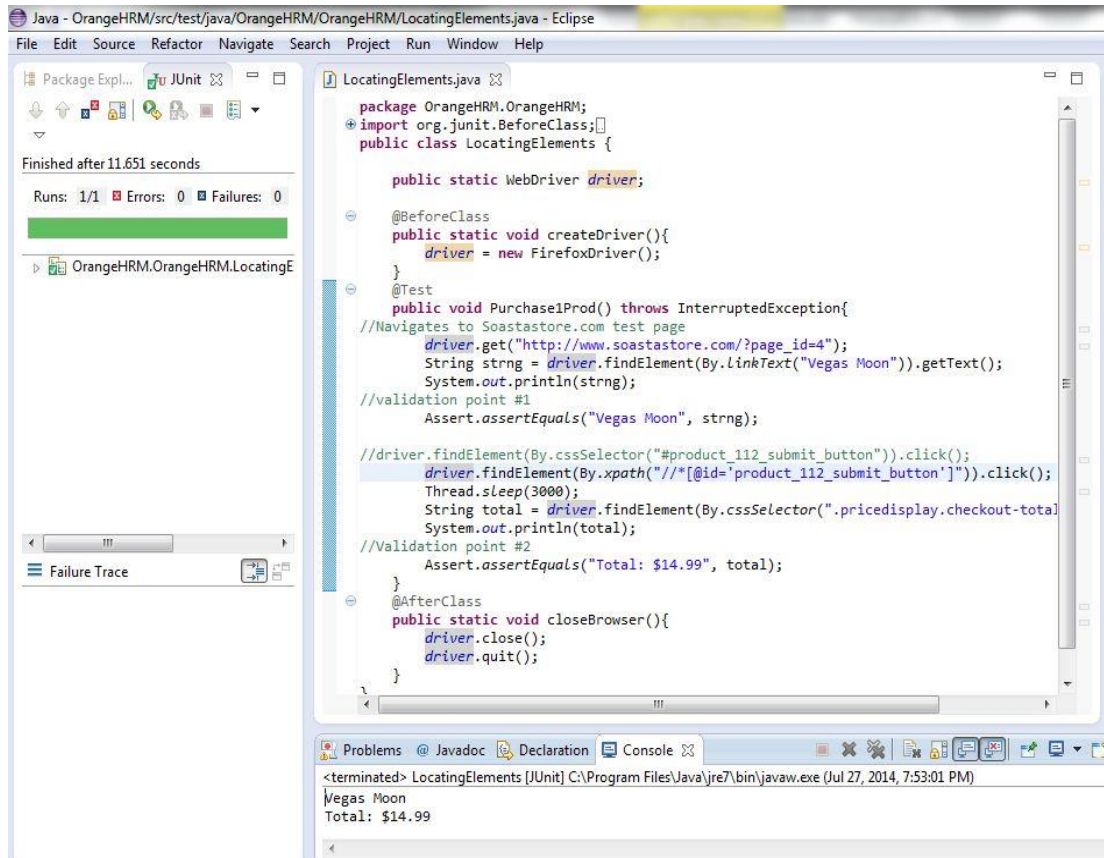


Figure 4.14

Automation of purchase a book from a web site to illustrate locating elements

This chapter discussed about some software testing tools mainly Selenium as a functional automation tool of web based applications. The automated software tools ease the operations of testing. With such tools, software testing can be done much faster than traditional testing methods. APPENDIX will show further extension of web based application automation with Selenium tool and other resources.

Chapter 5: Conclusion

Every software developer knows the importance of software testing. It ensures the software product does what it is supposed to do. One of the major advantages of software testing is that it eliminates or reduces errors leading to faster development time and less cost. Thus theoretically, the final product can be delivered on time with a reasonable price tag. Therefore testing is beneficial for the client as well as the developer.

Recent software project requirements and specifications rapidly change, in order to satisfy the customer it is necessary to deliver an error free product on time. To find at least all required functional errors the final product need to be tested properly. To maintain all of these in a given project, there have to have a great project management and to test the project properly, it is mandatory to have a STLC. There are so many methods, levels and types of software testing applied throughout the STLC. The main methods include static testing and dynamic testing. In static testing the code itself is tested without running the software whereas dynamic testing is done by executing the software itself. Both these methods can be applied for better results. White-box and black-box methodologies are another two different methods used for testing. White-box testing is applied when the internal procedures and operations of the software are known and black-box testing is when those are unknown. A combination of these two methods is known as gray-box testing and can be better than the other two methodologies alone under many circumstances. Software testing methods can broadly be classified as verification and validation. While verification answers the question 'Are we building the right thing?', validation focuses on the question 'Are we building the thing right?'. Main testing levels include unit testing, integration testing, system testing and acceptance

testing. These can be used at different stages of STLC. But all these levels of testing cannot be employed at all stages of the STLC. There are many test types that can be used for testing software. A set of these test types can be incorporated to test software. The types of tests used may depend of the software, the knowledge and experience of the testers, test schedules and deadlines etc. but theoretically no software can be tested 100%.

As stated earlier software testing eliminates or reduces errors leading to faster development time and lower cost. Unfortunately testing itself can be time consuming and hence costly. Valuable time that could be used for some other important thing has to be allocated for testing. Therefore, to enjoy the benefits of testing, it is important to reduce the time spent on testing without sacrificing the quality of the testing process. There are two major approaches to this end: one is to reduce time spent on testing by reducing the number of test cases, and the other, to automated repetitive test steps thereby increasing testing speed leading to reduced testing times.

Combinatorial testing aims at reducing the testing times. It achieves this by selecting a minimum number of test cases known as covering array depending on the number of test inputs. A reduction of test cases sometimes in the ranges of 100:1 leads to much less testing times and testing overheads. But the practical difficulty with combinatorial testing is preparing the covering array for systems with higher number of inputs. For this reason little attention was paid to combinatorial testing by the testers for almost a decade.

With the advent of new automated testing tools, software testing has become less arduous. Different automated software testing tools have emerged for testing different types of software. These have radically improved testing speeds and accuracy. For example here we

have demonstrated the Selenium tool. This wave of automated software testing tools has helped combinatorial testing too. Now the most difficult part in combinatorial testing - building the covering array - can be handed over to an automated tool. With this development, combinatorial testing should receive more attention as an efficient testing method.

Finally, this paper basically discussed and gave an introduction to the software testing with the aims and importance of software testing. Then it addressed two main problems of software testing and suggested two solutions in order to reduce the number of test cases and testing time using combinatorial testing and test automation accordingly. It is presented the solution to first problem using combinatorial testing while test automation providing solution to second problem in detail with some examples. In the Appendix it will be discussed, implementation of a framework for functional automation of an industrial standard web based software project by using basically Selenium and Java.

References

- [1] G. J. Myers and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, 2011.
- [2] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.
- [3] C. Kaner, “Exploratory testing,” in *Quality Assurance Institute Worldwide Annual Software Testing Conference*, 2006.
- [4] R. Kuhn, R. Kacker, Y. Lei, and J. Hunter, “Combinatorial software testing,” *Computer (Long. Beach. Calif.)*, vol. 42, no. 8, pp. 94–96, 2009.
- [5] P. R. Srivastava and K. Baby, “Automated software testing using metahurestic technique based on an ant colony optimization,” *Electron. Syst. Des. (ISED)*, 2010 Int. Symp., 2010.
- [6] Softwaretestinggenius.com, “Software testing genius,” 2012. [Online]. Available: <http://www.softwaretestinggenius.com/>. [Accessed: 19-May-2014].
- [7] K. Karhu, T. Repo, O. Taipale, and K. Smolander, “Empirical observations on software testing automation,” in *2009 International Conference on Software Testing Verification and Validation*, 2009, pp. 201–209.
- [8] T. Wissink and C. Amaro, “Successful test automation for software maintenance,” in *2006 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 265–266.
- [9] A. Zylberman and A. Shotten, “Test language - Introduction to keyword-driven testing,” *Methods & Tools*, 01-Jan-2010.
- [10] M. S. Milad Hanna Nahla El-Haggar, “A review of scripting techniques used in automated software testing,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 1, pp. 194–202, 2014.
- [11] Smartbear.com, “What is agile testing? | SmartBear software,” 2013. [Online]. Available: <http://smartbear.com/all-resources/articles/what-is-agile-testing/>. [Accessed: 24-May-2014].
- [12] E. Collins, A. Dias-Neto, and V. F. de Lucena Jr., “Strategies for agile software testing automation: An industrial experience,” in *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, 2012, pp. 440–445.

- [13] Netsolutionsindia.com, “How AGILE testing helps in building better software products |,” 2013. [Online]. Available: <http://www.netsolutionsindia.com/blog/agile-testing-for-software-products/>. [Accessed: 24-May-2014].
- [14] O. Tekin and G. B. Cetin, “Application test process in product life cycle,” in *2012 6th International Conference on Application of Information and Communication Technologies (AICT)*, 2012, pp. 1–6.
- [15] Softwaretestingfundamentals.com, “Software Testing life cycle (STLC) | software testing fundamentals,” 2012. [Online]. Available: <http://softwaretestingfundamentals.com/software-testing-life-cycle/>. [Accessed: 23-May-2014].
- [16] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in *Proceedings of the Future of Software Engineering at ICSE 2007*, 2007, pp. 85–103.
- [17] S. Zhang, D. Saff, Y. Bu, and M. D. Ernst, “Combined static and dynamic automated test generation,” *Proc. 2011 Int. Symp. Softw. Test. Anal. - ISSSTA '11*, no. 2, p. 353, 2011.
- [18] D. Balzarotti, M. Cova, V. Felmetzger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, “Saner: Composing static and dynamic analysis to validate sanitization in web applications,” in *Proceedings - IEEE Symposium on Security and Privacy*, 2008, pp. 387–401.
- [19] J. Lönnberg, A. Korhonen, and L. Malmi, “MVT,” in *Proceedings of the Working Conference on Advanced Visual Interfaces - AVI '04*, 2004, p. 385.
- [20] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat, “Classification of software testing tools based on the software testing methods,” in *2009 Second International Conference on Computer and Electrical Engineering*, 2009, pp. 229–233.
- [21] A. Sawant, P. H. Bari, and P. M. Chawan, “Software testing techniques and strategies,” *J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 980–986, 2012.
- [22] A. Arcuri, M. Z. Iqbal, and B. Lionel, “Black-box system testing of real-time embedded systems using random and search-based testing,” *Test. Softw. Syst.*, vol. 6435, pp. 95–110, 2010.
- [23] A. MacCormack, “Product-development practices that work: How internet companies build software | MIT Sloan management review,” *MIT Sloan Management Review*, 2001.
- [24] E. V Veenendaal, “Standard glossary of terms used in software testing, Version 1.2,” *Int. Softw. Test. Qualif. Board*, vol. 1, pp. 1–51, 2010.

- [25] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*. CRC Press, 2013.
- [26] T. Shiba, T. Tsuchiya, and T. Kikuno, “Using artificial life techniques to generate test cases for combinatorial testing,” in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, 2004.
- [27] D. Blue, I. Segall, R. Tzoref-brill, and A. Zlotnick, “Interaction-based test-suite minimization,” in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 182–191.
- [28] D. Kuhn, R. Kacker, and Y. Lei, “Practical combinatorial testing,” *NIST Spec. Publ.*, 2010.
- [29] D. Huizinga and A. Kolawa, *Automated Defect Prevention: Best Practices in Software Management*. John Wiley & Sons, 2007.
- [30] A. Holmes and M. Kellogg, “Automating functional tests using selenium,” in *AGILE 2006 (AGILE’06)*, pp. 270–275.
- [31] N. Uppal and V. Chopra, “Design and implementation in selenium IDE with web driver,” *Int. J. Comput. Appl.*, vol. 46, no. 12, pp. 8–11.
- [32] D. Burns, *Selenium 2 Testing Tools: Beginner’s Guide*. Packt Publishing, Ltd, 2012.
- [33] A. Bruns, A. Kornstadt, and D. Wichmann, “Web application tests with selenium,” *IEEE Softw.*, vol. 26, no. 5, pp. 88–91, Sep. 2009.
- [34] R. A. Razak and F. R. Fahrurazi, “Agile testing with selenium,” in *2011 Malaysian Conference in Software Engineering*, 2011, pp. 217–219.
- [35] F. Wang and W. Du, “A test automation framework based on WEB,” in *Proceedings - 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, ICIS 2012*, 2012, pp. 683–687.
- [36] L. Kong and Z. Yin, “The extension of the unit testing tool junit for special testings,” in *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS’06)*, 2006, vol. 2, pp. 410–415.
- [37] N. Javvaji, A. Sathiyaseelan, and U. Selvan, “Data driven automation testing of web application using selenium,” *isqtinternational.com*.
- [38] S. Company, *Maven: The Definitive Guide: The Definitive Guide*, vol. 4. O’Reilly Media, Inc., 2008.

Appendix

Test Automation Project with Eclipse and IntelliJ-Idea IDEs

Setup the environment to implement the project

Step-1: Install the Mozilla Firefox web browser (if required).

Step-2: Open Firefox and go to Tools menu → Add-ons, install Firebug and FirePath and then restart the browser.

Step-3: Go to <http://www.seleniumhq.org/download/> and install latest version of Selenium IDE as plug-in to Firefox web browser and restart it.

Step-4: Install the Java development kit and setup the environment variables.

Project implementation with the Eclipse IDE

Step-1: Download and install eclipse IDE

Step-2: Install maven build tool (In eclipse IDE go to Help → Install New Software...)

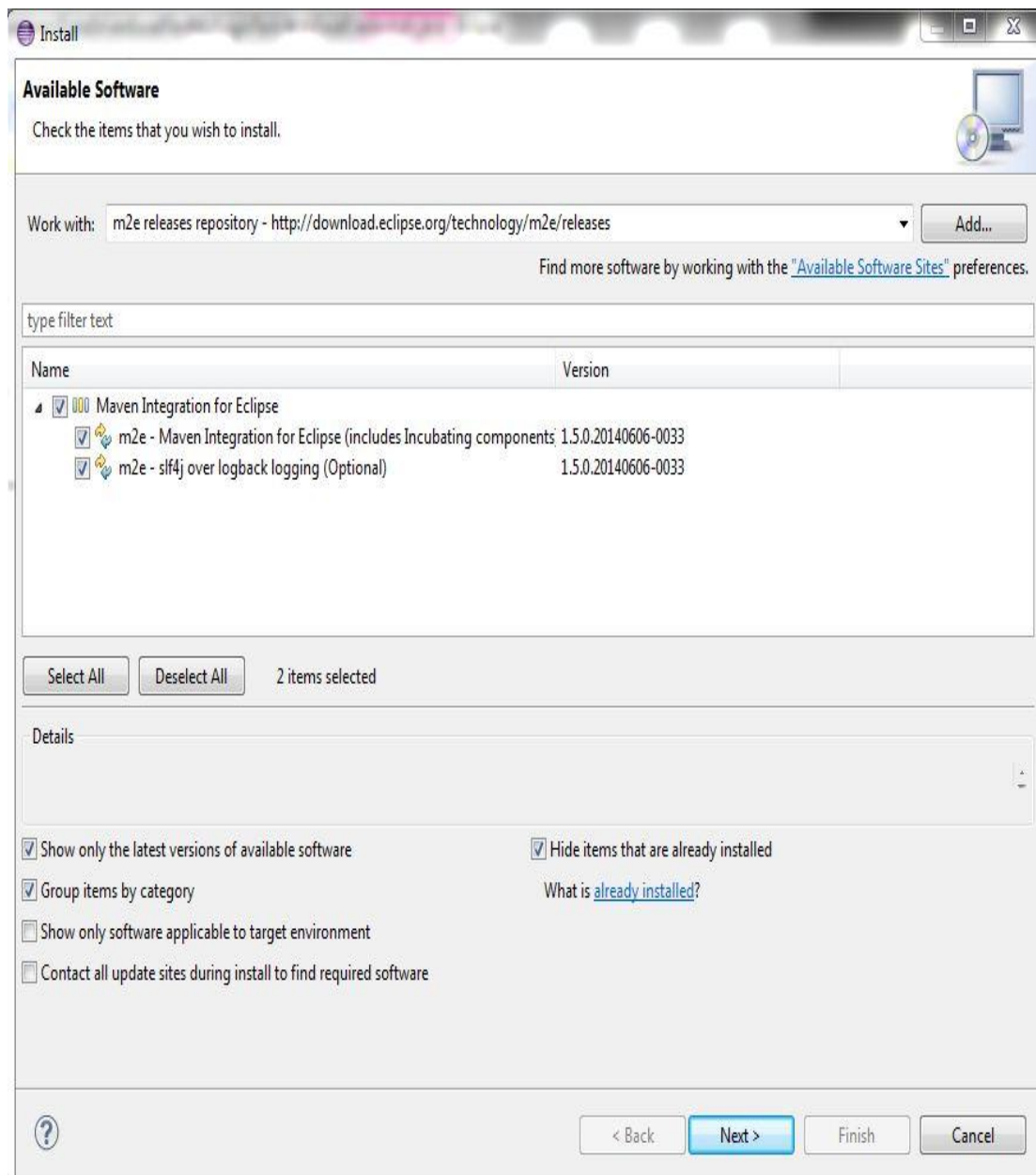


Figure 1

Installation of Maven build tool with the Eclipse IDE

Step-3: Install TestNG testing framework (In eclipse IDE go to Help → Install New Software...)

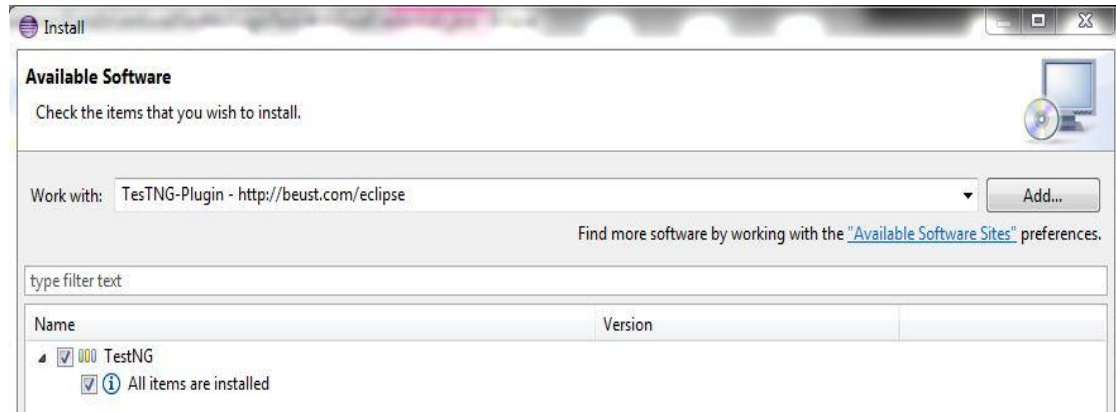


Figure 2

Installation of TestNG testing framework with the Eclipse IDE

Step-4: Write java code and add the properties, resources, test data files to the project

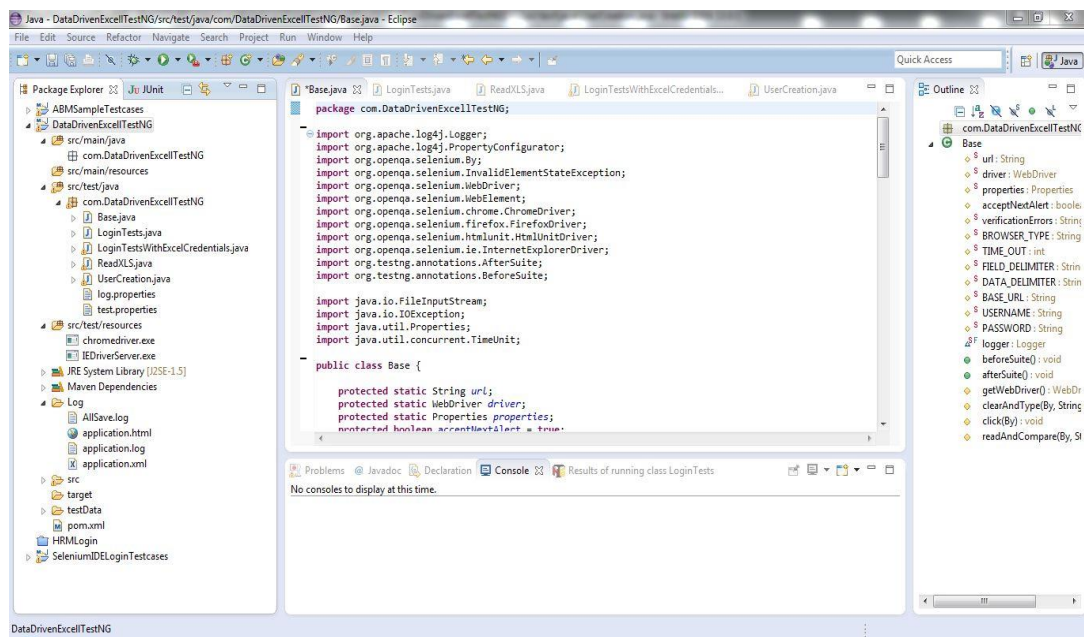


Figure 3

Project implementation with the Eclipse IDE

Step-5: Configure Maven POM file by adding required dependencies

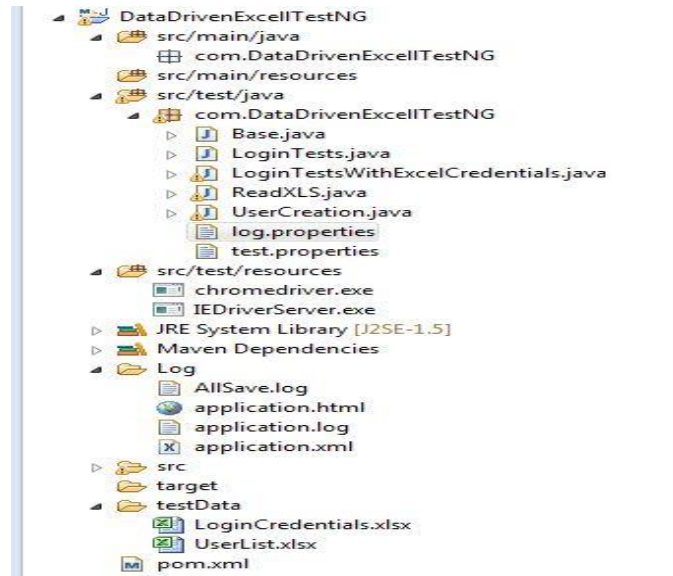


Figure 4

Project package explorer window with the Eclipse IDE

Step-6: Execute the project and see the results

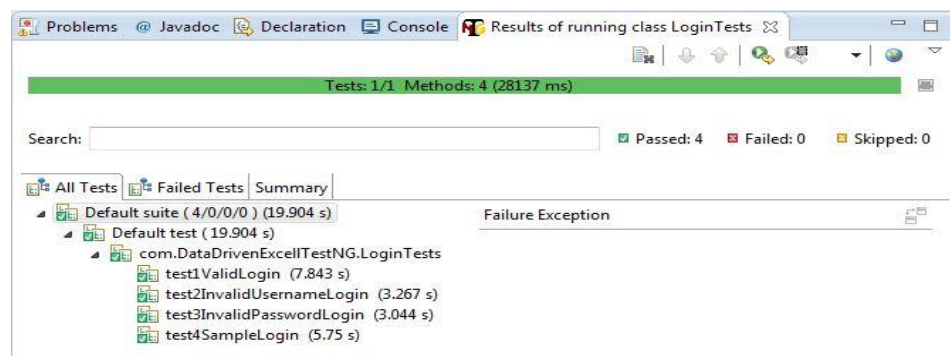


Figure 5

Test result of LoginTest.java class with the Eclipse IDE

Project implementation with the IntelliJ-IDEA IDE

Step-1: Download and install IntelliJ IDEA IDE

Step-2: Maven build tool and TestNG framework are pre-installed with IDEA IDE

Step-3: Write java code and add properties, resources, test data files to the project

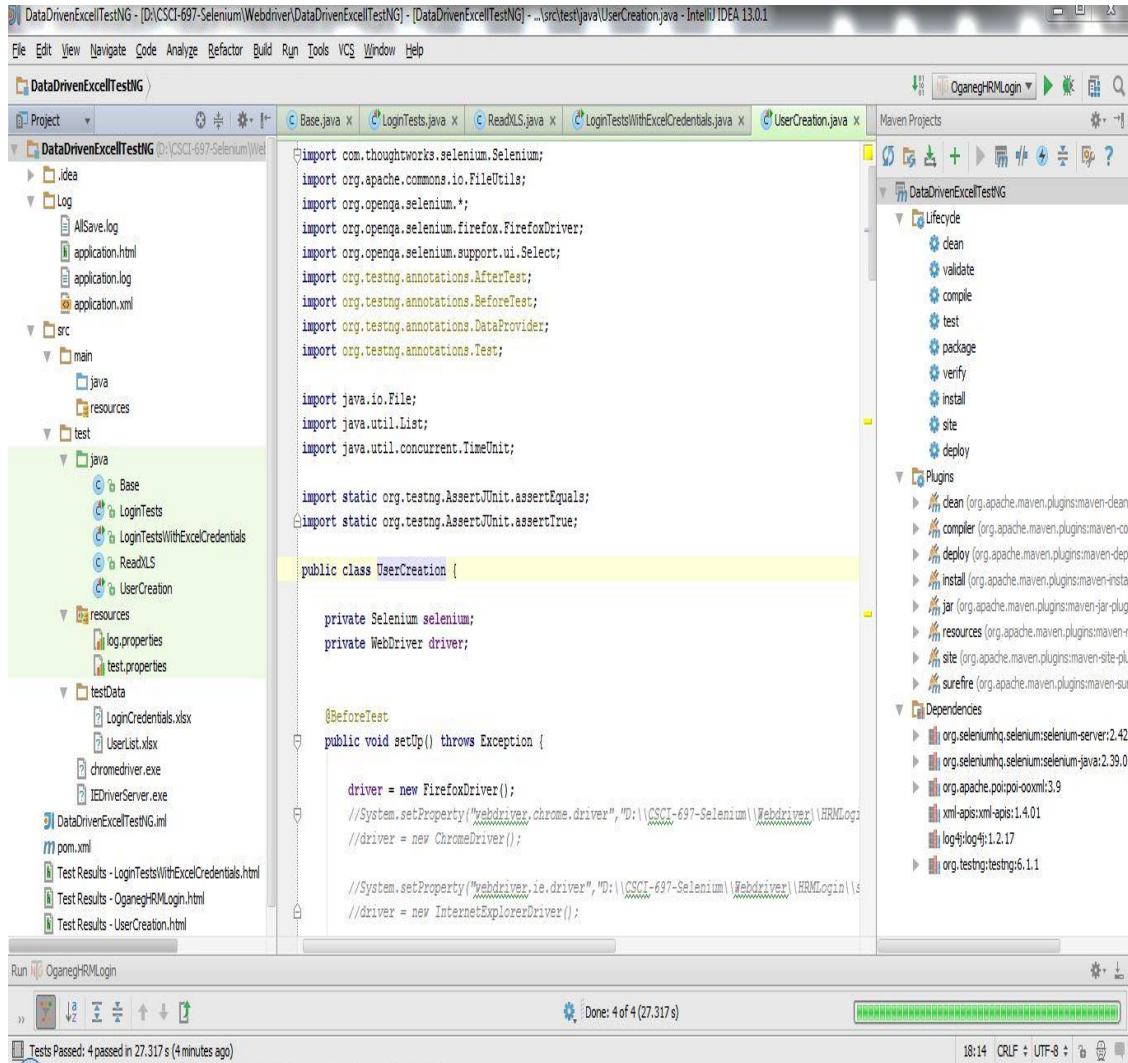


Figure 6

Implementation with the IDEA IDE

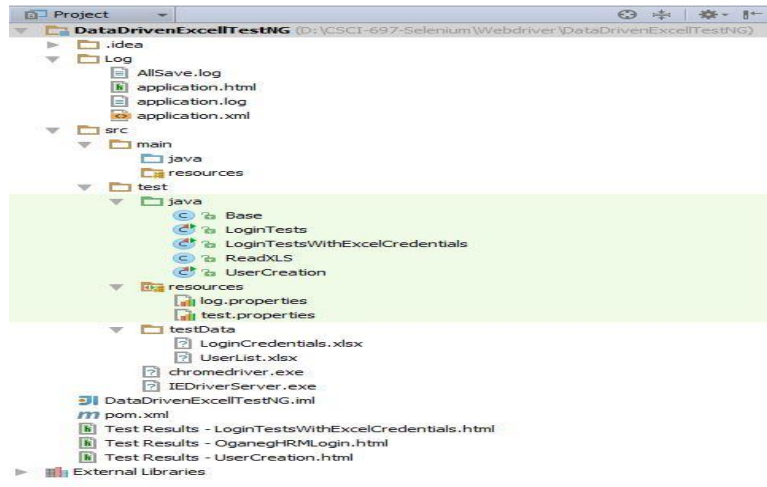


Figure 7

Project package explorer window with the IDEA IDE

Step-4: Configure Maven POM file by adding required dependencies

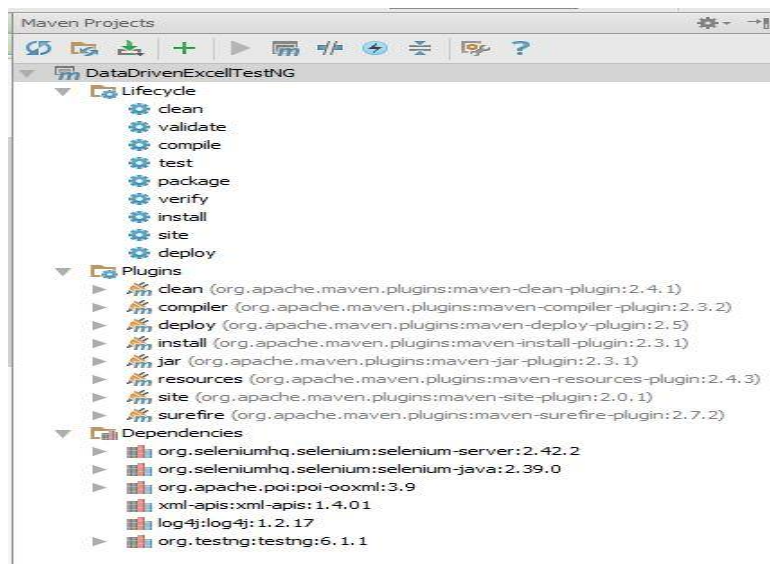


Figure 8

Maven projectwindow with the IDEA IDE

Step-5: Execute the project and see the results.

No Package	18.92 s
Login Tests	18.92 s
test1ValidLogin	passed 7.35 s
test2InvalidUsernameLogin	passed 3.35 s
test3InvalidPasswordLogin	passed 3.01 s
test4SampleLogin	passed 5.22 s
expected: <Welcome Admin> but was: <Welcome Catherine>	
Generated by IntelliJ IDEA on 8/9/14 10:43 PM	

Figure 9

Test result of LoginTest.java class with the IDEA IDE generated by HTML

LoginTestsWithExcelCredentials: 2 total, 2 passed	12.04 s
Collapse Expand	
No Package	12.04 s
LoginTestsWithExcelCredentials	12.04 s
testOrangeHRMLogin ("Admin", "admin")	passed 7.38 s
testOrangeHRMLogin ("admin", "admin")	passed 4.66 s
Generated by IntelliJ IDEA on 8/9/14 10:58 PM	

Figure 10

Test result of LoginTestsWithExcelCredentials.java class with the IDEA IDE



Figure 11

Test result of UserCreation.Java class with the IDEA IDE generated by HTML

Java Source Code of Multiple Test Classes

To implement all the comment calls: Base.Java

```
package com.DataDrivenExcellTestNG;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.openqa.selenium.By;
import org.openqa.selenium.InvalidStateException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeSuite;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class Base {

    protected static String url;
    protected static WebDriver driver;
    protected static Properties properties;
    protected static boolean acceptNextAlert = true;
    protected static StringBuffer verificationErrors = new StringBuffer();

    protected static String BROWSER_TYPE;
    protected static int TIME_OUT;
    protected static String FIELD_DELIMITER;
    protected static String DATA_DELIMITER;

    protected static String BASE_URL;
    protected static String USERNAME;
    protected static String PASSWORD;
    static final Logger logger = Logger.getLogger(Base.class);

    @BeforeSuite
    public void beforeSuite() throws Exception {
        properties = new Properties();
        try {
```

```

properties.load(new
FileInputStream("C:/Users/Akalanka/TestAutomation/DataDrivenExcellTestNG/src/test/
java/com/DataDrivenExcellTestNG/test.properties"));
PropertyConfigurator.configure("C:/Users/Akalanka/TestAutomation/DataDrivenExcellT
estNG/src/test/java/com/DataDrivenExcellTestNG/log.properties");

BROWSER_TYPE = properties.getProperty("BROWSER_TYPE");
TIME_OUT = Integer.parseInt(properties.getProperty("TIME_OUT"));
FIELD_DELIMITER = properties.getProperty("FIELD_DELIMITER");
DATA_DELIMITER = properties.getProperty("DATA_DELIMITER");

BASE_URL = properties.getProperty("BASE_URL");
USERNAME = properties.getProperty("USERNAME");
PASSWORD = properties.getProperty("PASSWORD");

driver = getWebDriver();

    } catch (IOException e) {
Logger.fatal(e);
System.exit(0);
    }
}

@AfterSuite
public void afterSuite() throws Exception {
driver.quit();
}
protected WebDriver getWebDriver() throws Exception {

if (BROWSER_TYPE == null) {
Logger.info("BROWSER_TYPE not set. Starting default type");
driver = new HtmlUnitDriver();
    } elseif (BROWSER_TYPE.equalsIgnoreCase("Firefox")) {
driver = new FirefoxDriver();
    } elseif (BROWSER_TYPE.equalsIgnoreCase("Chrome")) {
System.setProperty("webdriver.chrome.driver", "C:/Users/Akalanka/TestAutomat
ion/DataDrivenExcellTestNG/src/test/resources/chromedriver.exe");
driver = new ChromeDriver();
    } elseif (BROWSER_TYPE.equalsIgnoreCase("InternetExplorer")) {
System.setProperty("webdriver.ie.driver", "C:/Users/Akalanka/TestAutomation/
DataDrivenExcellTestNG/src/test/resources/IEDriverServer.exe");
driver = new InternetExplorerDriver();
    } elseif (BROWSER_TYPE.equalsIgnoreCase("HtmlUnit")) {
driver = new HtmlUnitDriver();
    } else {
Logger.info("BROWSER_TYPE not identified. Starting default type");
driver = new HtmlUnitDriver();
    }

Logger.info("BROWSER_TYPE=" + BROWSER_TYPE);
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(TIME_OUT, TimeUnit.SECONDS);
return driver;
}

```



```

    }

    protected void clearAndType(By by, String text) throws Exception {
        WebElement field = driver.findElement(by);
        try {
            field.clear();
            field.sendKeys(text);
        } catch (InvalidElementStateException e) {
            logger.info(e.toString());
        }
    }

    protected void click(By by) throws Exception {
        driver.findElement(by).click();
    }

    protected void readAndCompare(By by, String text) throws Exception {
        WebElement field = driver.findElement(by);
        if (!field.getText().equals(text)) {
            verificationErrors.append("\nElement: ");
            verificationErrors.append(by.toString());
            verificationErrors.append("\nExpected Text: ");
            verificationErrors.append(text);
            verificationErrors.append("\nActual Text: ");
            verificationErrors.append(field.getAttribute("value"));
        }
    }
}

```

TEST-1: LoginTests.Java

```

package com.DataDrivenExcellTestNG;

import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;

public class LoginTests extends Base {

    By userName = By.id("txtUsername");
    By password = By.id("txtPassword");
    By subButton = By.id("btnLogin");
    By welcomeId = By.id("welcome");

    @Test
    public void test1ValidLogin() throws Exception {

        driver.get(BASE_URL + "symfony/web/index.php/auth/login");
    }
}

```



```

clearAndType(userName, "Admin");
clearAndType(password, "admin");
click(subButton);
readAndCompare(welcomeId, "Welcome Admin");
}

```

```

@Test
public void test2InvalidUsernameLogin() throws Exception {
    driver.get(BASE_URL + "symfony/web/index.php/auth/login");
    String userName = "abc123";
    driver.findElement(By.cssSelector("span.form-hint")).click();
    driver.findElement(By.id("txtUsername")).clear();
    driver.findElement(By.id("txtUsername")).sendKeys(userName);
    driver.findElement(By.id("txtPassword")).clear();
    driver.findElement(By.id("txtPassword")).sendKeys("admin");
    driver.findElement(By.id("btnLogin")).click();
    try {
        Assert.assertEquals("Retry Login",
            driver.findElement(By.cssSelector("h1")).getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }
    Assert.assertEquals("OrangeHRM", driver.getTitle());
}

```

```

@Test
public void test3InvalidPasswordLogin() throws Exception {
    driver.get(BASE_URL + "symfony/web/index.php/auth/login");
    String userPassword = "abc123";
    driver.findElement(By.id("txtUsername")).clear();
    driver.findElement(By.id("txtUsername")).sendKeys("Admin");
    driver.findElement(By.id("txtPassword")).clear();
    driver.findElement(By.id("txtPassword")).sendKeys(userPassword);
    try {
        Assert.assertEquals("( Username : Admin | Password : admin )",
            driver.findElement(By.cssSelector("font")).getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }

    try {
        Assert.assertEquals("LOGIN Panel",
            driver.findElement(By.id("logInPanelHeading")).getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }
    driver.findElement(By.id("btnLogin")).click();
}

```

```

@Test
public void test4SampleLogin() {
    driver.get(BASE_URL + "symfony/web/index.php/auth/login");
    driver.findElement(By.xpath("//div[@id='divUsername']/input")).clear();
    driver.findElement(By.xpath("//div[@id='divUsername']/input")).sendKeys("Admin");
    driver.findElement(By.xpath("//div[@id='divPassword']/input")).clear();
}

```

```

driver.findElement(By.xpath("//div[@id='divPassword']/input")).sendKeys("admin");
driver.findElement(By.xpath("//div[@id='divLoginButton']/input")).click();
driver.findElement(By.xpath("//div[@id='branding']/a[2]")).click();
try {
Assert.assertEquals("Welcome Catherina",
driver.findElement(By.xpath("//div[@id='branding']/a[2]")).getText());

        } catch (Error e) {
System.out.println(e.getMessage());
        }
driver.findElement(By.xpath("//div[@id='welcome-menu']/ul/li/a")).click();

    }
}

```

To read data from MS-Excel files: ReadXLS.java

```

package com.DataDrivenExcellTestNG;

import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ReadXLS {

    String excelPath;

    public ReadXLS(String path){
        excelPath = path;
    }

    public List<String> getData() {

        List<String> dataList = new ArrayList<>();
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(new File(excelPath));
            XSSFWorkbook workbook = new XSSFWorkbook(fis);
            XSSFSheet sheet = workbook.getSheet("TestData");

```

```

java.util.Iterator rows = sheet.rowIterator();

while (rows.hasNext()) {
XSSFRow row = ((XSSFRow) rows.next());
// int r=row.getRowNum();
java.util.Iterator cells = row.cellIterator();
int i = 0;
String[] testData= new String[3];
while (cells.hasNext()) {

XSSFCell cell = (XSSFCell) cells.next();
String value = cell.getStringCellValue();
if (!value.equals(null)) {
testData [i] = value;
i++;
        }
    }
    dataList.add(testData);
}

catch (Exception e) {
e.printStackTrace();
}
return dataList;
}

public List getDataFromAdminTest(int numberOfColumns) {
List dataList = new ArrayList();
FileInputStream fis = null;
try {
fis = new FileInputStream(new File(excelPath));
XSSFWorkbook workbook = new XSSFWorkbook(fis);
//get data from Test Data sheet of work book
XSSFSheet sheet = workbook.getSheet("Test Data");
Iterator rows = sheet.rowIterator();

while (rows.hasNext()) {
XSSFRow row = ((XSSFRow) rows.next());
Iterator cells = row.cellIterator();
int i = 0;
String[] testData= new String[numberOfColumns];
while (cells.hasNext()) {

XSSFCell cell = (XSSFCell) cells.next();
String value = cell.getStringCellValue();
if (!value.equals(null)) {
testData [i] = value;
i++;
        }
    }
    dataList.add(testData);
}
}

```

```

    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return dataList;
}
}

```

TEST-2: LoginTestsWithExcelCredentials.Java

```

package com.DataDrivenExcellTestNG;

import org.openqa.selenium.By;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import java.util.List;

public class LoginTestsWithExcelCredentials extends Base {

    By userName = By.id("txtUsername");
    By password = By.id("txtPassword");
    By subButton = By.id("btnLogin");
    By welcomeId = By.id("welcome");

    @Test(dataProvider = "myTest")
    public void testOrangeHRMLogin(String un, String pwd) throws Exception {

        driver.get(BASE_URL + "symfony/web/index.php/auth/login");
        clearAndType(userName, un);
        clearAndType(password, pwd);
        click(subButton);
        readAndCompare(welcomeId, "Welcome Admin");
    }

    @DataProvider(name = "myTest")
    public Object[][] getDataFromExcell() {
        String path =
            "C:/Users/Akalanka/TestAutomation/DataDrivenExcellTestNG/testData/LoginCredentials.xlsx";
        ReadXLS readXls = new ReadXLS(path);
        List<List> dataList = readXls.getData();
        Object obj[][] = new Object[dataList.size()][2];
        for (int i = 0; i < dataList.size(); i++) {
            String[] test = (String[]) dataList.get(i);

```

```

        String un = test[0];
        String pwd = test[1];
obj[i][0] = un;
obj[i][1] = pwd;
    }
    return obj;
}
}

```

TEST-3: UserCreation.Java

```

package com.DataDrivenExcellTestNG;

import com.thoughtworks.selenium.Selenium;

import org.apache.commons.io.FileUtils;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.Select;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import java.io.File;
import java.util.List;
import java.util.concurrent.TimeUnit;

import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;

public class UserCreation {

    private Selenium selenium;
    private WebDriver driver;

    @BeforeTest
    public void setUp() throws Exception {

        driver = new FirefoxDriver();
        //System.setProperty("webdriver.chrome.driver", "C:/Users/Akalanka/TestAutomation/DataDrivenExcellTestNG/src/test/resources/chromedriver.exe");
        //driver = new ChromeDriver();
    }

```

```

//System.setProperty("webdriver.ie.driver","C:/Users/Akalanka/TestAutomation/DataD
rivenExcellTestNG/src/test/resources/IEDriverServer.exe");
//driver = new InternetExplorerDriver();

        String baseUrl = "http://admin.netexam.com/";
selenium = newWebDriverBackedSelenium(driver, baseUrl);
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
siteLogin();
    }

@Test (priority = 0 ,dataProvider = "myTest")
publicvoidnetexamCreateNewUser(String runMode,StringfirstName ,String lastName
,String txtUserName ,String email ) throws Exception {

createNewUser(runMode,firstName,lastName,txtUserName,email);
    }

privatevoidsiteLogin() {
driver.get("http://admin.netexam.com/login.asp");
driver.findElement(By.id("email")).clear();
driver.findElement(By.id("email")).sendKeys("testing@netexam.com");
driver.findElement(By.id("password")).clear();
driver.findElement(By.id("password")).sendKeys("y7efhuh84t");
driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();
selenium.waitForPageToLoad("30000");

assertTrue(driver.findElement(By.cssSelector("BODY")).getText().matches("^\\s\\s\\s
*Welcome netexam testing[\\s\\s]*$"));
System.out.println("Site Login Successful ");
    }

privatevoidcreateNewUser(String runMode,StringfirstName ,String lastName ,String
txtUserName ,String email) throwsInterruptedException {

selenium.open("http://admin.netexam.com/user-addedit.asp");
selenium.waitForPageToLoad("90000");

Thread.sleep(500);
driver.findElement(By.id("firstname")).clear();
driver.findElement(By.id("firstname")).sendKeys(firstName);
driver.findElement(By.id("lastname")).clear();
driver.findElement(By.id("lastname")).sendKeys(lastName);
driver.findElement(By.id("txtUsername")).clear();
driver.findElement(By.id("txtUsername")).sendKeys(txtUserName);
driver.findElement(By.id("businessemail")).clear();
driver.findElement(By.id("businessemail")).sendKeys(email);

Thread.sleep(500);
new
Select(driver.findElement(By.id("cmbTimeZone"))).selectByVisibleText("(GMT+06:00)
Astrana, Dhaka");

```

```

new Select(driver.findElement(By.id("Select3"))).selectByVisibleText("United States");
new Select(driver.findElement(By.id("Select4"))).selectByVisibleText("English");

Thread.sleep(500);
driver.findElement(By.cssSelector("#show-btn-Courses")).click();
driver.findElement(By.xpath("//div/table/tbody/tr/td/table/tbody/tr/td/div/input")).sendKeys("v6Testgroup");
driver.findElement(By.xpath("//img[2]")).click();
Thread.sleep(500);
driver.findElement(By.xpath("//div[2]/div/div/div[2]/div/div/table/tbody/tr/td[3]/div")).click();
driver.findElement(By.xpath("//td[2]/table/tbody/tr/td/table/tbody/tr/td/table/tbody/tr[2]/td[2]/em/button")).click();

Thread.sleep(500);
new Select(driver.findElement(By.id("Select5"))).selectByVisibleText("Default");
new Select(driver.findElement(By.id("Select7"))).selectByVisibleText("Default");

Thread.sleep(500);
driver.findElement(By.cssSelector("#Submit2")).click();

selenium.waitForPageToLoad("30000");

try {
    File scrFile =
    ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
    FileUtils.copyFile(scrFile, new
        File("C:\\Automation\\Netexam\\screenshots\\user_profile_new.png"));
}
catch (Exception e) {
    e.printStackTrace();
}

assertEquals("Add/Edit a User: Akalanka Mailewa",
driver.findElement(By.cssSelector("td.ArialBlack15")).getText());

System.out.println("User Creation Successful");
}

private void createDuplicateUser() throws InterruptedException {
    selenium.open("http://admin.netexam.com/user-addedit.asp");
    selenium.waitForPageToLoad("90000");

    Thread.sleep(500);
    driver.findElement(By.id("firstname")).clear();
    driver.findElement(By.id("firstname")).sendKeys("Akalanka1");
    driver.findElement(By.id("lastname")).clear();
    driver.findElement(By.id("lastname")).sendKeys("Mailewa");
    driver.findElement(By.id("txtUsername")).clear();
    driver.findElement(By.id("txtUsername")).sendKeys("Akalanka002");
    driver.findElement(By.id("businessemail")).clear();
    driver.findElement(By.id("businessemail")).sendKeys("Akalanka002@yahoo.com");
}

```

```

Thread.sleep(500);
new
Select(driver.findElement(By.id("cmbTimeZone"))).selectByVisibleText("(GMT+06:00)
Astrana, Dhaka");
new Select(driver.findElement(By.id("Select3"))).selectByVisibleText("United
States");
new Select(driver.findElement(By.id("Select4"))).selectByVisibleText("English");

Thread.sleep(500);
driver.findElement(By.cssSelector("#show-btn-Courses")).click();
driver.findElement(By.xpath("//div/table/tbody/tr/td/table/tbody/tr/td/div/input")
).sendKeys("v6Testgroup");
driver.findElement(By.xpath("//img[2]")).click();
Thread.sleep(500);
driver.findElement(By.xpath("//div[2]/div/div/div[2]/div/div/table/tbody/tr/td[3]/
div")).click();
driver.findElement(By.xpath("//td[2]/table/tbody/tr/td/table/tbody/tr/td/table/tbo
dy/tr[2]/td[2]/em/button")).click();

Thread.sleep(500);
new Select(driver.findElement(By.id("Select5"))).selectByVisibleText("Default");
new Select(driver.findElement(By.id("Select7"))).selectByVisibleText("Default");

Thread.sleep(500);
driver.findElement(By.cssSelector("#Submit2")).click();

selenium.waitForPageToLoad("30000");

try {
    File scrFile =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(scrFile, new
    File("C:\\Automation\\Netexam\\screenshots\\user_profile_duplicate.png"));
}
catch (Exception e) {
e.printStackTrace();
}

assertEquals("The new Username you entered already exists.\nThe email you entered
already exists.", driver.findElement(By.cssSelector("span.wrong")).getText());
System.out.println("Duplicate User Detection Successful");
}

@AfterTest
public void tearDown() throws Exception {
selenium.stop();
}

//-----get data from Admin Test Cases Excel file

```



```

@DataProvider(name = "myTest")
public Object[][] getDataFromExcell() {
    String path =
"C:/Users/Akalanka/TestAutomation/DataDrivenExcellTestNG/testData/UserList.xlsx";
    int numberOfColumns = 5 ; // number of columns in excel sheet
    ReadXLSreadXls = new ReadXLS(path);
    List<dataList> = readXls.getDataFromAdminTest(numberOfColumns);
    Object obj[][] = new Object[dataList.size()][numberOfColumns];
    for (int i = 0; i < dataList.size(); i++) {
        String[] test = (String[]) dataList.get(i);
        String runMode = test[0];
        String firstName = test[1];
        String lastName = test[2];
        String txtUserName = test[3];
        String email = test[4];

        obj[i][0] = runMode;
        obj[i][1] = firstName;
        obj[i][2] = lastName;
        obj[i][3] = txtUserName;
        obj[i][4] = email;
    }
    return obj;
}
}

```

Project properties files

1. test.properties

```

#Firefox, Chrome, InternetExplorer, HtmlUnit
BROWSER_TYPE=Firefox
TIME_OUT=10
FIELD_DELIMITER =,
DATA_DELIMITER =#

BASE_URL=http://demo.orangehrmlive.com/
USERNAME=Admin
PASSWORD=admin

```

2. log.properties

```

# Rolling File Appender
log4j.appender.R=org.apache.log4j.RollingFileAppender
# Path and file name to store the log file.
log4j.appender.R.File=C:/Users/Akalanka/workspace/HRMLogin/Log/AllSave.log

```

```
log4j.appender.R.MaxFileSize=500KB
```

```
#####
```

```
# Define the root logger with file appender
log4j.rootLogger= DEBUG, FILE
# Console appender configuration
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
# Pattern to output the caller\u2019s file name and line number.
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p[%t](%F:%L)\u2013m%n
```

```
# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=./Log/application.log

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern= %d{yyyy-MM-dd}-%t-%x-%-5p-%-10c:%m%n
```

```
#####
```

```
# Define the root logger with file appender
##log4j.rootLogger = DEBUG, HTML
```

```
#Define the file appender
##log4j.appender.HTML=org.apache.log4j.FileAppender
##log4j.appender.HTML.File=./Log/application.html
```

```
# Define the html layout for file appender
##log4j.appender.HTML.layout=org.apache.log4j.HTMLLayout
##log4j.appender.HTML.layout.Title=Application logs
##log4j.appender.HTML.layout.LocationInfo=true
##log4j.appender.HTML.Threshold=DEBUG
```

```
#####
```

```
# Define the root logger with file appender
##log4j.rootLogger = DEBUG, XML
```

```
#Define the file appender
##log4j.appender.XML =org.apache.log4j.FileAppender
##log4j.appender.XML.File=./Log/application.xml
```

```
# Define the html layout for file appender
##log4j.appender.XML.layout =org.apache.log4j.xml.XMLLayout
```

Project resources

- chromedriver.exe
- IEDriverServer.exe

Project log files

- AllSave.log
- application.log
- application.html
- application.xml

Project test data files

- LoginCredentials.xlsx
- UserList.xlsx

Test result files generated in IDEA IDE

- Test Results - LoginTestsWithExcelCredentials.html
- Test Results - LoginTests.html
- Test Results - UserCreation.html

Apache maven project object model (POM) file

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.SCSU.abm</groupId>
  <artifactId>DataDrivenExcellTestNG</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>DataDrivenExcellTestNG</name>
  <description>DataDrivenExcellTestNG</description>

  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-server</artifactId>
      <version>2.42.2</version>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>2.39.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.poi</groupId>
      <artifactId>poi-ooxml</artifactId>
      <version>3.9</version>
    </dependency>
    <dependency>
      <groupId>xml-apis</groupId>
      <artifactId>xml-apis</artifactId>
      <version>1.4.01</version>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
```

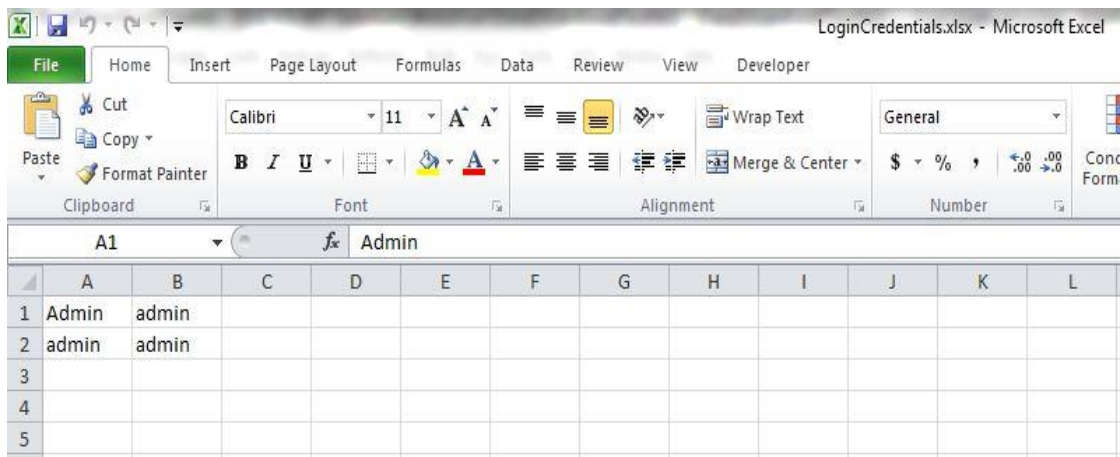
```

<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>6.1.1</version>
</dependency>
</dependencies>

</project>

```

Sample test data files created with MS-Excel



	A	B	C	D	E	F	G	H	I	J	K	L
1	Admin	admin										
2	admin	admin										
3												
4												
5												

Figure 12

Login information stored with a MS-Excel file

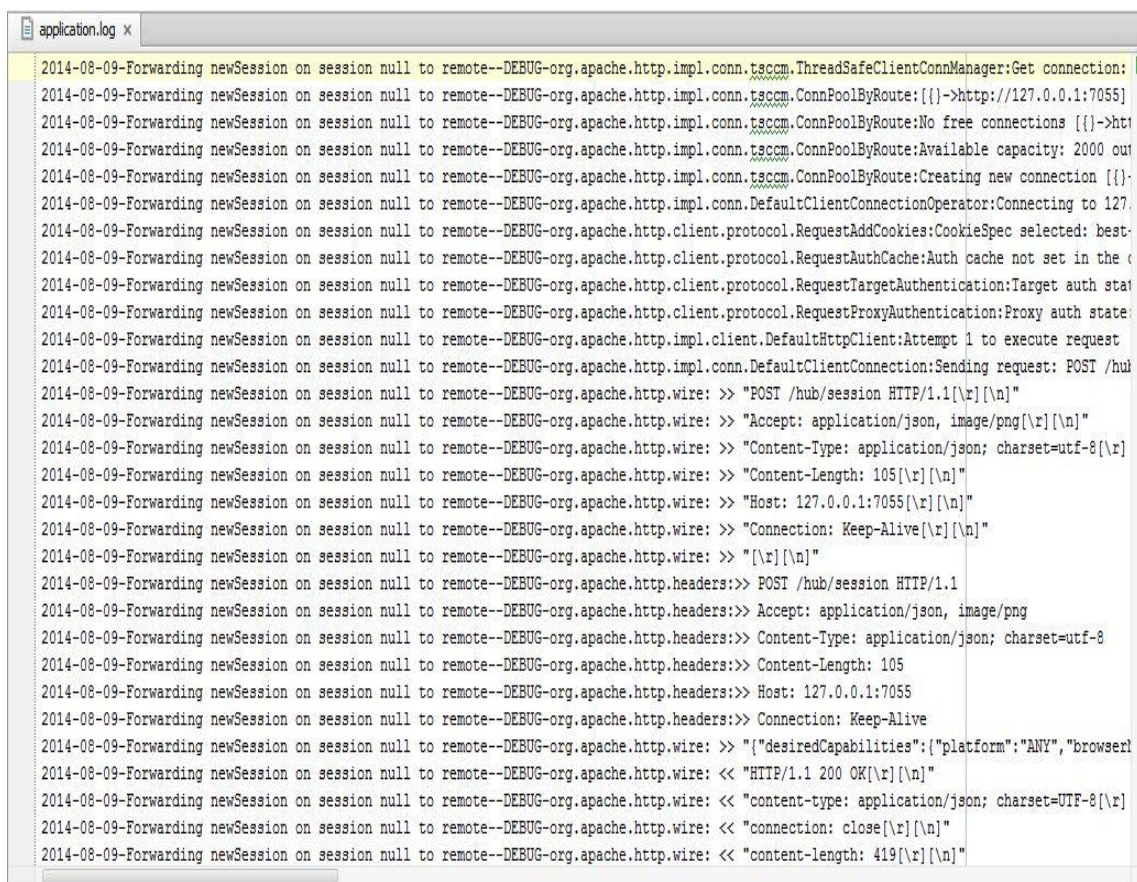
The screenshot shows the Microsoft Excel interface with the 'Home' tab selected. The ribbon includes options for Clipboard, Font, Alignment, and Number. The active cell is E1, which contains the formula `=First001@yahoo.com`. The spreadsheet data is as follows:

	A	B	C	D	E	F	G	H	I
1	F	First	Test	First001	First001@yahoo.com				
2	F1	First1	Test1	First1002	First1002@yahoo.com				
3									
4									

Figure 13

User creation information stored with a MS-Excel file

Log file generated by the project



```

application.log x
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager:Get connection:
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.tsccm.ConnPoolByRoute:[]->http://127.0.0.1:7055
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.tsccm.ConnPoolByRoute:No free connections []->ht
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.tsccm.ConnPoolByRoute:Available capacity: 2000 out
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.tsccm.ConnPoolByRoute:Creating new connection []-
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.DefaultClientConnectionOperator:Connecting to 127.
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.client.protocol.RequestAddCookies:CookieSpec selected: best-
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.client.protocol.RequestAuthCache:Auth cache not set in the
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.client.protocol.RequestTargetAuthentication:Target auth stat
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.client.protocol.RequestProxyAuthentication:Proxy auth state
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.client.DefaultHttpClient:Attempt 1 to execute request
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.impl.conn.DefaultClientConnection:Sending request: POST /hub
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "POST /hub/session HTTP/1.1[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "Accept: application/json, image/png[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "Content-Type: application/json; charset=utf-8[\r]
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "Content-Length: 105[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "Host: 127.0.0.1:7055[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "Connection: Keep-Alive[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> POST /hub/session HTTP/1.1
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> Accept: application/json, image/png
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> Content-Type: application/json; charset=utf-8
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> Content-Length: 105
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> Host: 127.0.0.1:7055
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.headers:>> Connection: Keep-Alive
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: >> "{\"desiredCapabilities\":{\"platform\":\"ANY\",\"browserI
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: << "HTTP/1.1 200 OK[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: << "content-type: application/json; charset=UTF-8[\r]
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: << "connection: close[\r]\n"
2014-08-09-Forwarding newSession on session null to remote--DEBUG-org.apache.http.wire: << "content-length: 419[\r]\n"

```

Figure 14

Application log file generated by the project

Finally it can be concluded that the APENDIX discussed in detail the implementation of a framework for functional automation of a real world industrial standard web based software project by using basically Selenium and Java with some other software tools and IDEs for different web browsers.