

St. Cloud State University theRepository at St. Cloud State

Culminating Projects in Mechanical and
Manufacturing Engineering

Department of Mechanical and Manufacturing
Engineering

1-2017

Optimizing Software Quality through Automation Testing

Ankit Sharma

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

Recommended Citation

Sharma, Ankit, "Optimizing Software Quality through Automation Testing" (2017). *Culminating Projects in Mechanical and Manufacturing Engineering*. 65.

https://repository.stcloudstate.edu/mme_etds/65

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

Optimizing Software Quality through Automation Testing

by

Ankit Sharma

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in partial Fulfillment of the Requirements

for the Degree

Master of Engineering Management

December, 2016

Starred Paper Committee:

Ben Baliga, Chairperson

Hiral Shah

Balasubramanian Kasi

Abstract

The current business application is large, multi-tiered, distributed and integrated which require higher level of sophistication to implement and manage. The current quality methodologies rely on manual work which makes the application venerable due to its limitation and entails higher cost. Running complete regression suite manually every time is cumbersome and often do not complete due to either time or resource limitation. Finding more defects during testing life cycle has tremendous effect on the quality of an application. This project is intended to run more number of tests in lesser time and reduction in overall cost of the project and this has been achieved by implementing an automation tool. Various tools and frameworks are studied to fulfill this requirement, also the results are stated and compared. The implication of implementing an automation test tool is higher software quality assurance.

Acknowledgments

This project completion is mere impossible without the assistance and valuable guidance from many individuals.

I am thankful to Dr. Hiral Shah, Associate Professor for Engineering Management Program at St Cloud State University. Her support, guidance, and assistance assisted me to complete this project successfully.

I would take this opportunity to thank Dr. Ben Baliga, Professor and Graduate Director for Engineering Management Program at St. Cloud State University for his support and guidance.

Also, I would like to thank Prof. Kasi Balasubramanian for serving on the committee and support throughout the project.

Table of Contents

	Page
List of Tables	6
List of Figures	7
Chapter	
I. Introduction	9
Introduction	9
Problem Statement	10
Nature and Significance of the Problem	10
Object of the Project	11
Project Questions	11
Limitations	11
Summary	12
II. Background and Review of Literature	13
Introduction	13
Background Related to Problem	13
Literature Related to Methodology	15
Summary	23
III. Methodology	24
Introduction	24
Design of Study	24
Data Collection	42

	5
Chapter	Page
Summary	42
IV. Data Analysis	43
Introduction	43
Data Analysis	43
Data Presentation	50
Summary	51
V. Results, Conclusion, and Recommendations	52
Introduction	52
Results	52
Conclusion	54
Recommendations	54
References	56

List of Tables

Table	Page
1. Comparison Study of Automation Tools	25
2. Scope of Testing	31
3. Test Tools	33
4. Defect Priority	34
5. Testing Team	35
6. Test Schedule	35
7. Approval and Sign Off	36
8. Sample Test Case	41
9. Sample Test Script	41
10. Manual vs Automation Comparison	50

List of Figures

Figure	Page
1. Relative Cost to Fix Software Defects	14
2. Manual vs Automated Test Comparison	14
3. Expanded Testing Model	17
4. Software Testing Life Cycle	19
5. End to End Testing Process	30
6. Test Phases	32
7. Java Installation	36
8. Eclipse Download Page	37
9. Configuring Eclipse IDE	38
10. Download Selenium Java Client	39
11. Configuring Build Path	39
12. Add External Jars	40
13. Running TestNG Script	45
14. Refresh Page	46
15. Generated Test Output Folder	46
16. Index.html Report	47
17. Default Test Report	48
18. Email-able Report	49
19. XSLT Report	50
20. Manual vs Automation Time Study	52

Figure

Page

21. Total Cost of Testing 53

Chapter I: INTRODUCTION

Introduction

Software drives the competitive businesses in this global economy. Software quality is the key, and cost of defects late in the life cycle become prohibitive [1].

With greater complexity from technology, software sourcing, compliance, etc., addressing defects is becoming a challenging job. Companies focusing on traditional quality approaches are dealing with some level of bugs post-release. The Systems Sciences Institute at IBM has reported that “the cost to fix an error found after product release was four to five times as much as one uncovered during design, and up to 100 times more than one identified in the maintenance phase.”

The Current Web Application who is owned by Comcast who is a leading provider of communications, entertainment and cable product and services. With up to 6TB of Internet traffic per second per each day, the Comcast network handles

- Over 142 million completed phone calls,
- Over 136 million delivered emails,
- Over 12 million received voicemails [2].

To handle this complex system, it requires sophisticated software to streamline the business. Bugs in the later stages of software lead to huge business losses (monetary and customer trust). Therefore, it becomes very important to ensure quality pre-product release and must have sufficient confidence over the product to prevent any adverse situation.

Problem Statement

The current business application is large, multi-tiered, distributed and integrated. It requires higher level of sophistication to implement and manage. The current quality methodologies rely on manual software quality assurance which makes the application venerable due to its limitations. Manual Testing Process is very cumbersome and complete regression tests are almost impossible to implement. The cost involved in assuring quality of application is very high due to large number of man hours. Moreover, manual tests are not accurate due to human error and more time consuming, hence it is less reliable.

It is hard to test and very expensive in case of urgent patch fixes to production application during overnight and weekends. Testing time is directly proportion to the number of test cases. With increasing competition, companies are facing pressure to release newer products sooner in the market. Due to time limitations, often the quality of product is compromised.

Nature and Significance of the Problem

One of the major problem in introducing new software product or making changes to the existing one is testing time. Test teams spend most of their time running test cases, it takes as much as a day just to test one new feature of a system and often test fails due to system time outs. Full regression tests have been so expensive and team avoid whenever possible. Needless to say, execution is manual. The turnaround time for releasing a new version of software after it has been sufficiently tested is too long and seems to be ever increasing. The test team is busy

doing manual testing instead of producing new test specifications, or updating old one to match new ones to match the new requirement. Consequently, test documentation is lagging.

Objective of the Project

1. To Assure Quality of software by running more tests in lesser time.
2. To Gain confidence in the application.
3. Reduce man hours to decrease overall cost.
4. Run regression suite during weekends and overnight.

Project Questions

Following project questions will be answered after the implementation of automation process:

1. How much is the total reduction in turnaround time of completing regression test suite?
2. What are the total savings due to automation?
3. How the automation testing tool is selected?

Limitations

- Automation vs Manual study comparison compares one-time static cost only.
- No of product releases (major, minor) are assumed based on historical data.

Summary

This chapter briefly discusses the nature and significance of the existing problem, how it largely reduces the quality of the software and increase the overall cost.

Objectives and project questions are also discussed here. The next chapter covers the literature background knowledge related to the project.

Chapter II: BACKGROUND AND REVIEW OF LITERATURE

Introduction

This chapter focuses on reviewing the background related to problem, literature related to methodology that has been implemented to solve the problem. Also, we will briefly discuss about the company background and issues related to our existing problem.

Background Related to Problem

Comcast is an American global mass media conglomerate and is the largest broadcasting and cable television company in the world by revenue. Comcast services U.S. residential and commercial customers in 40 states and the District of Columbia [3]. Comcast Cable is one of the nation's largest video, high-speed Internet and phone providers to residential customers under the XFINITY brand and provides these services to businesses. Comcast has invested in technology to build a sophisticated network that delivers the fastest broadband speeds, and brings customers personalized video, communications and home management offerings.

It offers a wide variety of products and services to its customers, few of them are listed here:

- A) Xfinity TV,
- B) Xfinity Internet,
- C) Xfinity Voice,
- D) Xfinity Home,

etc. It is currently servicing more than 800,000 customers.

Serving the needs of large customers require a sophisticated application, capable of handling large data or traffic maturely and gracefully. The current Web Application owned by Comcast, PA is a cross browser, cross platform application. Due to its vast features and complexity it is not easy to test an entire application before any major release. Nevertheless, quality is being compromised. The Science institute of IBM has reported that the cost to fix an error after product release was four to five as much as one uncovered during design, and up to 100 time more than one identified during maintenance phase.

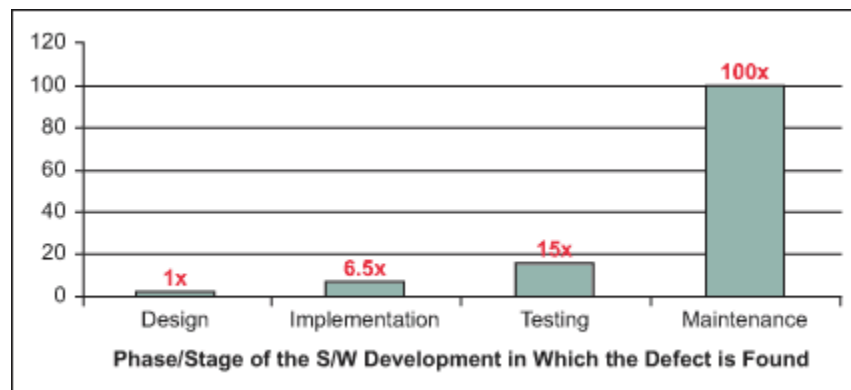


Figure 1: Relative Cost to Fix Software Defects

The following graph illustrate the relationship between time and cost of testing:

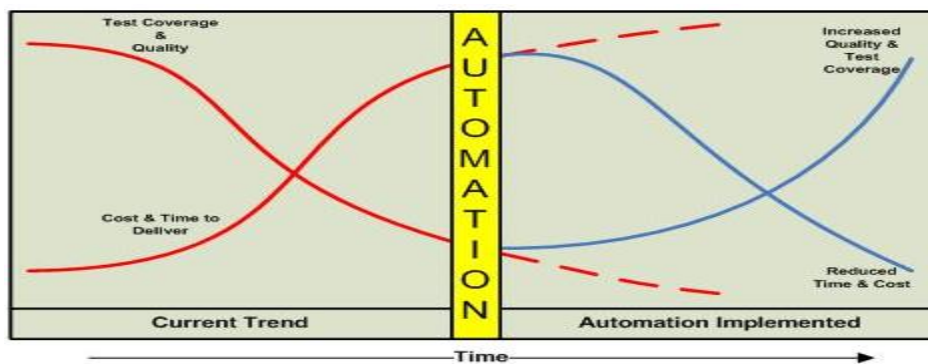


Figure 2: Manual vs Automated Test Comparison

Manual Efforts provide a smaller test coverage area and compromised overall quality of the product and hence the goal is to reduce the time of testing and the cost of delivery, while increasing test coverage and quality [4].

It is also evident that resources to test are mostly freely available during nights or weekends. However, running tests during nights and weekends are not feasible and most of the times surpass the project budget. Also, reduces the morale and performance of the team as they need to work over weekends or nights.

It is very important to differentiate between the test that execute very few times to the frequent ones. It is well worth automating only those tests that execute many times and are among the best candidate for the regression suites.

The current application test execution is manual and its regression suite consists of 480 test cases. Averaging 12 min per test execution, total time require completing testing is $480 * 12 = 5760$ Minutes or 96 hours. I.e. Team of 3 persons working 8 hours/day require 4 days to complete. Looking at these numbers it proves that it is good candidate for automation Testing rather than Manual.

Literature Related to Methodology

Nowadays with booming technology, Software is becoming essential part of human life from any mobile application (e.g., banking app) to medical appliance (e.g., life support system). Many people have experienced with software which do not work as anticipated. This kind of Software that does not work properly could result in lot of problems including time, money loss, some consequences may also lead to loss of business or reputation, in many cases it can be devastating and could cause damage

to human life or death. Therefore, it is necessary to test any software while it is in development stage and before final operational use.

The main objective of Software testing is to ensure the system/software under development is functioning correctly as per specifications and is bug or fault free. Bug can be of any type of error which produces incorrect results or catastrophic malfunction. So, to risk of problems occurring after software is implemented live, rigorous testing is necessary. Software testing also ensures the quality of product by increasing software's reliability which helps to gain customers confidence. The aim of any software project is to deliver software as per the customer provided specifications. That means project will be successful if:

- The customer needs should be specified correctly.
- And the developed software must meet that specifications exactly. The customer also wants the software to be delivered within given budget and timeline. Quality of product is also directly proportional to the maintenance cost.

A model of testing. Programming testing includes more than bolstering contributions to a program and watching comes about. Programming today additionally has states and communicates with put away information and the PC environment. Figure 3 reproductions the information sources and results for any product. Such a model is essential in test mechanization since it gives classifications to distinguish the information sources and results that must be checked and controlled amid computerized testing. For even a straightforward mechanized test

that sustains contributions to the SUT, the computerized test ought to confirm the normal direct results. On the off chance that the program should change the framework environment or any information sets, then some confirmation ought to be performed to affirm the right environment and information values after the test. At the point when analyzer computerizes tests he should guarantee that such issues are distinguished.

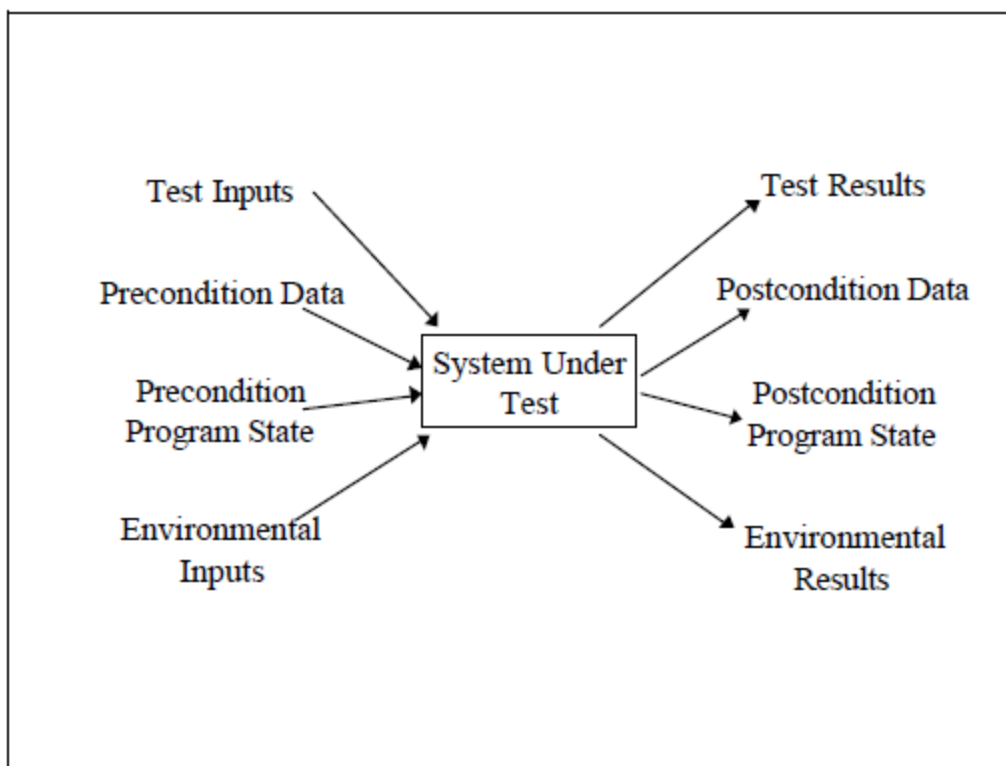


Figure 3: Expanded Testing Model (adapted from [5]).

Test levels. There are mainly four perceived levels of tests: unit/Component testing, Integration/reconciliation testing, System testing, and Acceptance testing. Tests are as often as possible assembled by where they are included the product improvement prepare, or by the level of specificity of the test.

1. **Unit Testing:** It searches for defects and verifies for functionality, of software (e.g., modules, programs, objects, classes, etc.) that are separately testable. It may be done in isolation from the rest of the system.
2. **Integration Testing:** It tests interfaces between components, interactions to different parts of a system, such as operating system, file system, hardware or interfaces between systems.
3. **System Testing:** It is concerned with the behavior of a whole system/product as defined by the scope of development project or program.
4. **Acceptance Testing:** The objective in Acceptance testing is to build up trust in the framework, parts of the framework or non-practical attributes of the framework. Discovering deformities is not the primary center in acknowledgment testing. Acknowledgment testing may evaluate the framework's status for sending and utilize, even though it is not the last level of testing. For instance, a substantial scale framework mix test may come after the Acceptance test for a framework.

Testing lifecycle. It comprises of set of activities which are completed in each manner to assure the quality of Software Under Test (SUT) Following are the activities which constitute the Software Testing Life Cycle (STLC). Every stage has predefined list of activities, set of Deliverables, Entry and Exit Criteria.

Figure 4 below shows different stages of STLC.

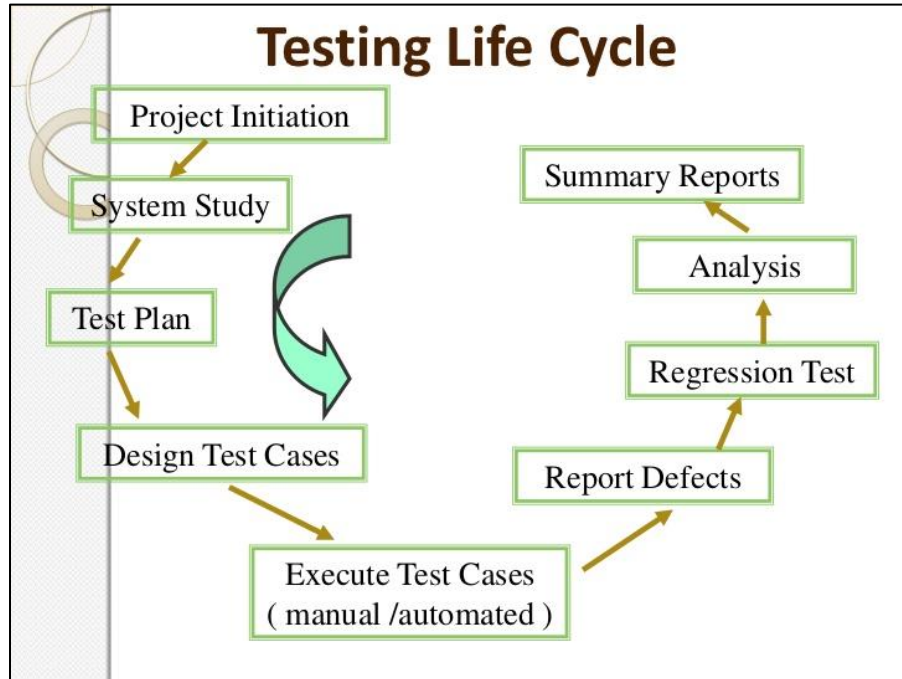


Figure 4: Software Testing Life Cycle

Manual vs. automation testing. Manual testing provides a way to quickly evaluate a product and provide testers with a familiarity of the features during the development process. Testers create test cases based on their ability to determine whether requirements are met. But initially these test cases should be executed manually, both for the sake of verification of the necessary steps, and to record test scripts for automation in the future [6].

Manual testing also includes exploratory testing, which enables testers to learn more about the application, while also identifying areas that potentially need more test cases to fully understand weaknesses and risk. Testers use exploratory testing to better understand weaknesses and determine which parts of the application need more test cases.

Lastly, manual testing is especially valuable early in the development of features and the user interface, as layouts and controls are often changing almost daily in response to design considerations and user feedback. It can be more time-consuming to maintain and change automated scripts than to execute tests manually.

From manual to automated process. Often Manual testing is defined as a state where a tester starts each test, relates with it, and understands, evaluates, and reports the results. While Automation Testing is when there is a tool for running of test cases without tester. Generally, test cases are said automated when all the following elements are present. If any of them is missing, then tests are considered as semi-automated. (Which is sometimes most cost-effective)

- Ability to run two or more number of test cases,
- Ability to run a subset of all the automated test cases,
- No involvement is needed after initiation the tests,
- Automatically sets-up and/or records the relevant test environment parameters,
- Runs the test cases,
- Captures the relevant results,
- Compares actual with expected results and flags differences,
- Analyzes and reports pass/fail for each test case and for the test run.

Key factors in automated testing. The underlying strides in while getting ready for test automation is to group and see some key variables about the Software Under Test (SUT): Identify what programming is to be tried, its segments and

elements we need to test, and the earth encompassing the SUT. These variables are basic to the mechanization engineering. Also, comprehend the current and accessible test product components and instruments for testing and test automation in the SUT's surroundings.

Albeit in some cases self-evident, it is frequently illuminating to formally portray what is it we need to test and recognize it from every other component in the framework. It is additionally imperative to recognize what things we believe are outside the extent of our mechanization or we don't mean to test. A few related applications and utilities with various interfaces may contain the SUT, perhaps notwithstanding running in various situations. Early choices on which parts to incorporate, which to prohibit, and which components are most critical can put authoritative limits on the automation tasks and generously diminish their many-sided quality.

Following those SUT components are finalized environment and interfaces must be recognized. The input and results data types are also imperative. Testing can succeed when the tester and automation tools perform a useful test and draw proper conclusions based on the results. Automation is valuable to augment the tester by performing tasks that are tedious or impossible for a human or are more cost effective to automate. Different types of tests are run at different times and not all related tests have to be run at one time.

The extent of the arranged automation undertakings likewise relies on upon the current and accessible test ware components and instruments. The test ware

components incorporate most the product, documentation, test cases, information, programs, and related methodology required for all the test exercises. The devices incorporate working framework utilities, test determination and control programs, examination schedules, and so forth, that are utilized to do the testing and automation. Two regions of test automation require uncommon consideration.

Result Capture can be a colossal assignment when you understand the universe of conceivable comes about because of running programming. The actual results of running a test are considerably more than survey data on the screen. Framework environment factors, memory and file contents, program status, messages, and so on, may all be affected by the right running of a program. Since we are searching for blunders, we should incorporate every one of the things that the program could affect—a much bigger arrangement of things we must check as "results."

Result Analysis is a second region that can spell inconvenience for an automation exertion. Things that will be checked expecting 'as Result' should be distinguished and one need vision what those Results ought to be. For manual testing analyzer takes a gander at the presentations, examine inward factors and program states, and fulfill that the test passed or failed. For automation testing analyzer, must program the automation apparatuses to play out similar errand. Physically, the analyzer chooses the arrangement of tests and examinations, and it fluctuates based upon the information they find.

There might be one test, however there might be a few "right" results from it, and there are an extensive number of ways a mistake would show itself. In an automated test environment analyzer, must arrangement the check of results with the goal that he know when the test passed and when it doesn't. That implies recognizing exceptionally critical signs of blunders and methodically checking them after each test

Summary

Test automation also has several roles in the development process. As testers grow more confident in his work, it saves the time and effort of repeating the same tests over the period. Recording these tests produces scripts that automated testing tools can execute at any time, often with different sets of data. Automation brings an important aspect to testing practices. It assures consistency in how a test is performed. Once the validity of a given test is established, automating it provides a defined benchmark of how the test is run on a regular basis.

Automation is also essential for regression testing, the practice of executing already-passed tests throughout the development process, to ensure that already-implemented features aren't broken as development continues. Regressions suites typically contain hundreds or even thousands of tests that can help maintain find issues throughout the development lifecycle [6].

Chapter III: METHODOLOGY

Introduction

Automation of software testing is like a software development process. It goes through the same life cycle as in the development of software product. The important think that must be taken care of who is writing the scripts. There is always a conflict on who writes the scripts whether a developer or a testing team member. It is always a good idea and normally followed by many organizations that the effort should be a collaborated effort between tester and the developer. The automation process goes through a lot of effort taking collaborated work because a lot of emphasis is given for the time and financial constraint.

Design of Study

The automation process may be divided into many phases but in general perspective goes through the following phases.

- I. **Test tool selection.** Automation testing success largely depends on the selection of right testing tool. Following comparison is drawn between different tools available in the market and based on the SUT (system under test) appropriate tool was selected.

Table 1: Comparison Study of Automation Tools

Sr. No.	Automation Tools				
	Eggplant Functional	iMacros	Selenium	Test Studio	Watir
Skilled resource to allocate for automation tasks?	No	No	Yes	No	Yes
Price	Free	Free browser add-on Standard: \$495 Enterprise: \$995 Web Browser Component: \$2,995	Free - Open Source	\$79/mo. up to \$2,999 one time	Free
Support All Web Browsers	Yes	No	Yes	Few	No
HTTPS	Yes	Yes	Yes	Yes	Yes
Is it suitable for the project environment and technology you are using?	No	No	Yes	No	Yes
Does the tool integrate with other testing tools like test planning and test management?	Yes	Yes	Yes	Yes	Yes
Support test data input from various data files such as Excel, XML, Text file etc.	Yes	No	Yes	Yes	Yes
Recorder	Yes	Yes	only Firefox	Yes	Yes

Further brain storming was done on the following points and based on the discussion; it was decided to go with the Selenium.

Following are the points that were discussed further to gain more insight about the tool and its appropriateness.

- Strong Skill of the team and ease of use.
- Environment Support.
- Support of multiple frameworks.
- Minimize training cost of the selected tool.
- Test Reports and Results.

II. **Test plan.** After studying the requirements of Software, detailed Test plan was created to describe the scope, approach and schedule of testing, etc.

1. Introduction

1.1 Objective

The objective of the Test Plan is to detail the approach, define responsibilities, define test deliverables and describe status reporting procedures to be employed for the testing phases of the XZ Games project. The test plan provides the framework used to plan and manage the testing effort. It seeks to outline the scope, schedules, responsibilities, resources, metrics, issues, risks and environment needs required to complete the testing.

The test plan supports the following objectives:

- Outline the components that will be tested in this release.
- Outline project test schedule and milestones.
- Outline the various phases of the testing process.
- Identify dependencies, assumptions and risks.
- Define the test environments that will support the various testing phases.
- Outline the roles and responsibilities related to the testing process.

- Detail the deliverables that will be produced during each phase of testing including test plan, test procedures and test execution metrics.
- Define the test data management process that will support the various testing phases.
- Establish an approach for full end-to-end testing.

1.2 Purpose of Sign-off

The sign-off this document indicates that the signing reviewers agree with the stated testing approach for the XZ Games project. The required signing reviewers are listed in the table in Section 8 below.

1.3 Project Background

This Project objective is to stream 20 video games to cable subscribers and allowing them to play games using Android and Apple smartphones and tablets as controllers. It primarily focuses on enhancing and refining this new gaming experience and making it available to many more customers soon.

The following test methodologies are planned for this project:

- Unit testing
- System testing
- Regression testing
- User Acceptance Testing
- Production checkout

Please refer to Section 3.1 of this master test plan for the definition of each test type.

2. Test Controls Verification and Validation

The deliverables for each test phase will be verified and validated through peer review. The major and more critical documents such as master test plan and detailed test plan will be reviewed with the different development teams.

2.1 Entry/Exit Criteria

Entry and exit criteria are a set of conditions that must be satisfied before entering or exiting a test phase. The criteria state what is required (for example, from previous phases) to support a given phase (entry criteria) and what is required of a given phase to determine completeness (exit criteria). Entry and exit criteria are defined for each phase to assure quality deliverables, and correct closure and handover, from one phase to the next.

Some exit criteria may satisfy the entry criteria of a phase other than the one next in line. The entry criteria for one phase will usually ensure that the exit criteria from the previous phase are completed, and will also include any additional set-up criteria. For instance, system test execution entry criteria may include the requirement that all assembly test exit criteria be met. An additional entry criterion to system test may be that the system test environment is established and all system test preparation exit criteria be met.

Please find the entry and exit criteria per test phase in Section 3 of this document.

2.2 General Prerequisites to the Test Phases

1. Business Requirement Document
2. Change Request Forms (if any)
3. Requirement Traceability Matrix
4. Software Requirement Specification Document
5. High Level Design Document
6. Low Level Design Document

2.3 General Test Deliverables

The test deliverables are as follows:

- Requirements traceability Matrix
- Test plan per phase
- Test cases
- Test automation scripts
- Test report per phase
- Defect or incident report

2.4 Test Methodology

This document addresses the validation component of the testing methodology.

The following diagram depicts the testing methodology to be used for the project/release. It is an end to end model developed by Software Development Technologies.

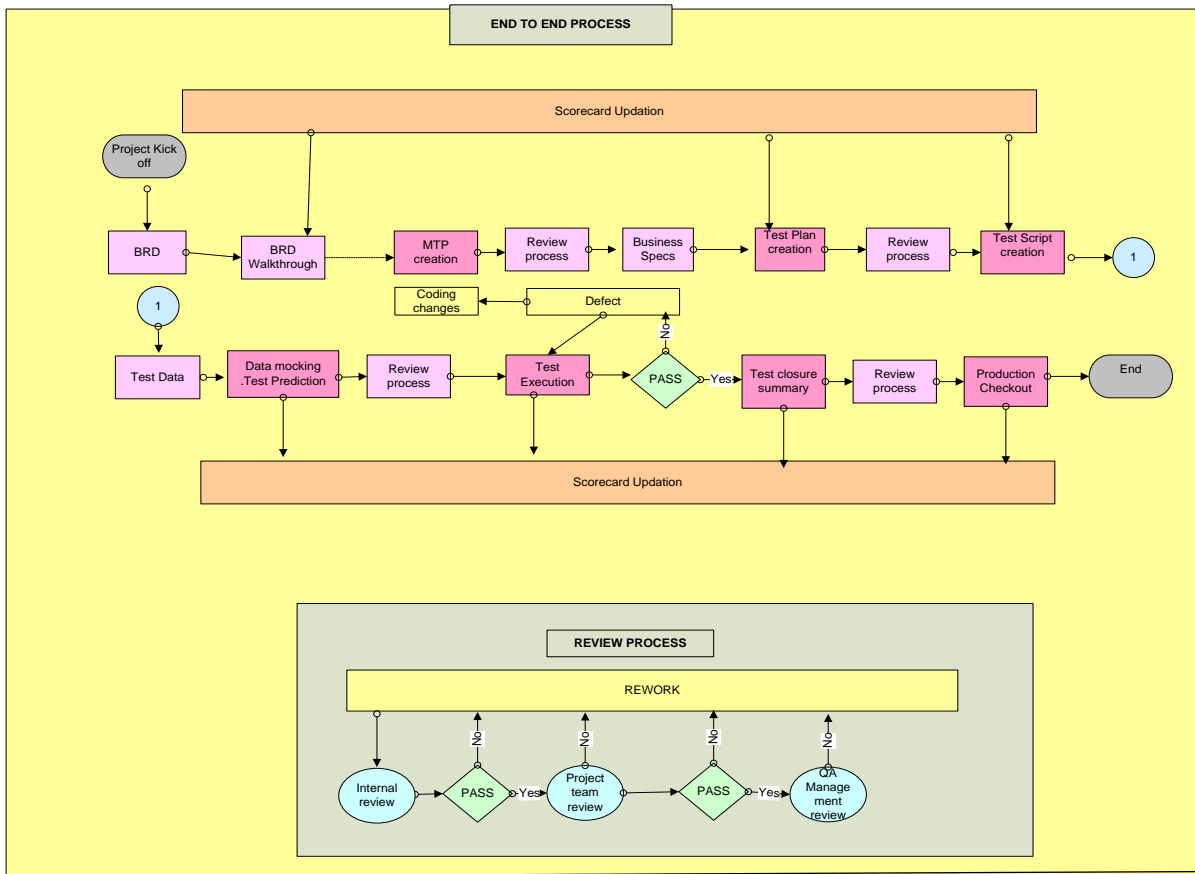


Figure 5: End to End Testing Process

3. Test Strategy

The following tests will be run during System/Regression Test Cycle (Iteration)

1. A secondary cycle will be utilized in the event of outstanding defects.

3.1 Scope of System/Regression Test:

Table 2: Scope of Testing

Sr. No.	Business Requirement	Detailed Specification
1	Pairing Page	<ol style="list-style-type: none"> 1. User image appears 2. Pairing code is present 3. Footer, footer text is present. 4. Submit button is present. 5. Privacy and Cookie Policy link is working etc.
2	Invalid Paring Page	<ol style="list-style-type: none"> 1. Validate the text Let's try that again. 2. InvalidPairingCode Validate the text Please check your pairing code and enter one more time. 3. PairingCodeExpired Validate the error code - XG-200. 4. InvalidPairingCode: Re-pair Validate if the Image displayed on the Page 5. InvalidPairingCode: Re-pairing Validate the Input Field is displayed 6. InvalidPairingCode: Re-pairing Validate the Submit Button is available
3	Select Profile	<ol style="list-style-type: none"> 1. Validate the User Image picture in Select Profile Page is displayed. 2. Validate the Text-Please select your XFINITY profile. 3. Validate Create Player button is present. 4. Validate Manage Players button is present
4	Create Player	<ol style="list-style-type: none"> 1. Validate the User Image is displayed. 2. Validate the Text-What is the new Player's name. 3. Validate Continue button is present. 4. Validate Cancel button is present. 5. CreateNewPlayer Continue Button takes user to SelectAvater page
5	Tablet Initiated Flow	<ol style="list-style-type: none"> 1. Validate if Welcome Back screen appears. 2. Validate if XFINITY Games Powered by EA logo is displayed 3. Validate if avatar is displayed 4. Validate if displayed avatar matches with that of the user. 5. Validate if welcome message is displayed with the correct username. 6. Validate if Continue Playing button is present 7. Validate if Select Another TV button is present

3.2 Out of Scope

This section lists items that are not in scope for this project.

- Any defects that are revealed through testing that are not due to this project.
- For more please refer to Business Requirement Document.

3.3 Assumptions

This section lists assumptions that are made specific to this project.

- Devices with different operating system like Android, iOS are available.
- Test Environment is available and ready.

3.4 Test Phases

The following diagram depicts the test phases planned for XZ Games:

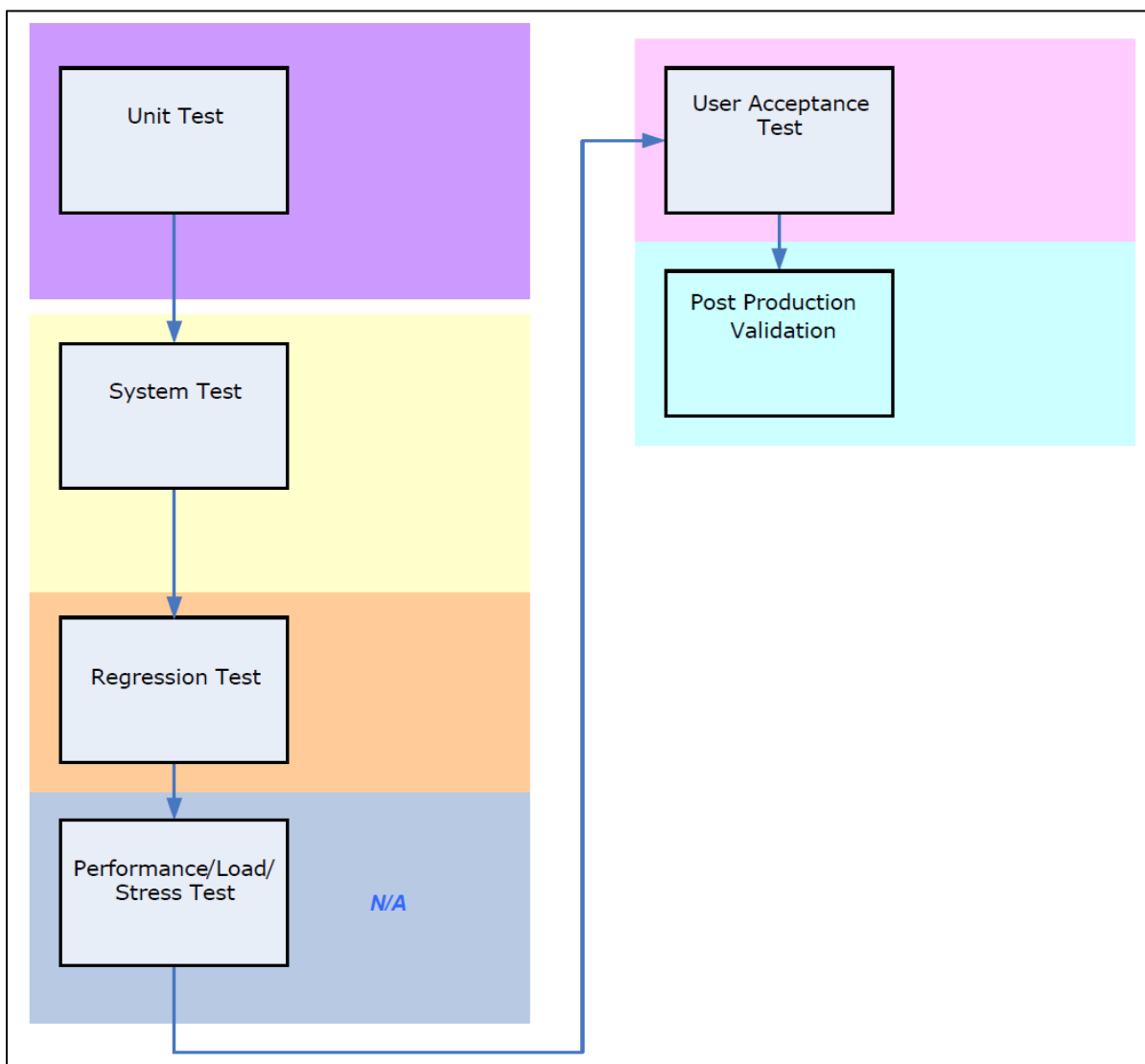


Figure 6: Test Phases

3.5 Success Criteria

A test cycle will be deemed as successfully tested when the cycle and associated test conditions have been executed with no defects arising. A test cycle will also be deemed as successfully tested even if a defect is raised during the execution of that cycle, if the defect is agreed by the project team and business owners as non-critical to the project release (low and medium priority). All critical and high priority defects must be successfully tested to proclaim the test cycle successful. A stakeholder's meeting will be conducted after the completion of full volume integration test to discuss any remaining defects. An assessment will be made at that time whether there are any defects that will risk the deployment schedule.

3.6 Test Tools

The following tools will be used in the entire test process:

Table 3: Test Tools

Tool Use	Tool Name
Test Plans	MS Word, MS Excel
Test Cases	MS Excel
Test Scripts	Eclipse IDE
Test Execution	Selenium, Appium, TestNG
Defect Tracking	JIRA

4. Defect Management

Defect management is the process of tracking and managing the discovery, resolution, and re-test of system defects identified during test execution. This process involves recording, reviewing, and prioritizing defects; assigning defects to

developers for fixing; and assigning testers to re-test fixes. It is essential to follow a process of this nature during test execution to ensure that all defects are recorded, resolved, and re-tested in a consistent and effective way, as quickly as possible. It is also essential to allow managers to accurately monitor the number, priority, and status of defects, so they can best manage the continued progress of the systems development project.

All defects will be logged JIRA.

4.1 Defect Priority

Defect priority descriptions are as follows:

Table 4: Defect Priority

Priority 1	LOW – Used to highlight minor defects that will be fixed only if time permits and does not impact the business' ability to use the application (e.g. cosmetic). Resolution could be in next release.
Priority 2	MEDIUM – Application generally functions, but needs to be fixed in the next release. Some piece of functionality fails. Business process needs modification to accommodate application behavior, but can wait for resolution. Need to fix in future build or release.
Priority 3	HIGH – Application can function using difficult workarounds. Must be fixed in next build or patch.
Priority 4	Very High – Used when there is a problem that significantly impacts the business' ability to use the application. Must be fixed before go-live with possible exception.
Priority 5	Urgent – Generally reserved for fatal errors which means that testing cannot continue without fix, and/or means that the service cannot go-live. Must be fixed before go-live.

5. Test Team Organization

Table 5: Testing Team

Name	Role	Responsibilities
Tester 1	Test Lead	<ul style="list-style-type: none"> • Study of BRD/SRS Document • Test Plan Creation • Test Scenarios Preparation in Test case sheet • Test cases review • Test Execution Result Review • Defect Review
Tester 2 Tester 3	Tester	<ul style="list-style-type: none"> • Test Scenarios Preparation in Test case sheet • Test cases creation, Test Script Creation • Test Execution

6. Test Schedule

The initial test schedule as below.

Table 6: Test Schedule

Task Name	Effort	Comments
Test Planning		
Review Requirements documents	2 d	
Create initial test estimates	1 d	
Create Test Plan	2 d	
Create Test Case, Scripts	12 d	
Staff and train new test resources	1 d	
Iteration 1 deploy to QA test environment	1 d	
System Testing 1	1 d	
Regression testing 2	1 d	
Iteration 2 deploy to QA test environment		
System testing 2	1 d	
Regression testing 2	1 d	
UAT	1 d	
Resolution of final defects and final build testing	1 d	
Deploy to Staging environment	1 d	
Release to Production	1d	

7. Mandatory Approval and Sign Off

Table 7: Approval and Sign Off

Name	Department/Team Role
PM	Project Manager
BA	Business Analyst Lead
SE1	Developer Lead
Tester 1	Test Lead

III. Environment Setup

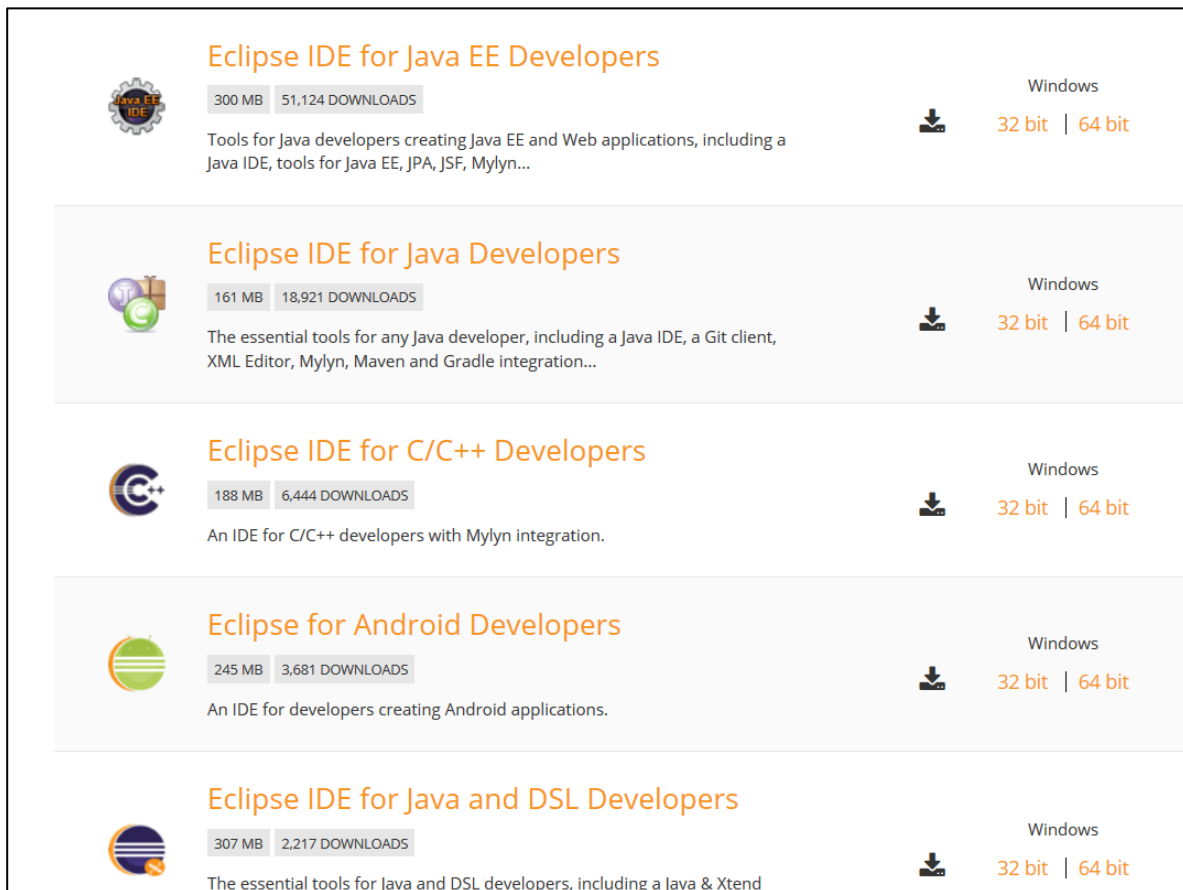
To develop and execute Selenium Web Driver Scripts, initial configuration need to be done. Setting up the environment requires the following steps.

Installing Java. In the first step, download and install JDK. Navigate to the following link <http://www.oracle.com/technetwork/java/javase/downloads/index.html> and install JDK by following the instructions.



Figure 7: Java Installation

Download eclipse. Eclipse is an IDE (integrated development Environment) and it is an open source software. Navigate to <http://www.eclipse.org/downloads/eclipse-packages/> and download the appropriate file.



The screenshot displays the Eclipse IDE download page, listing five different IDE packages for Windows. Each package includes an icon, size, download count, description, and download options for 32-bit and 64-bit systems.

Package Name	Size	Downloads	Description	Download Options
Eclipse IDE for Java EE Developers	300 MB	51,124	Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...	Windows 32 bit 64 bit
Eclipse IDE for Java Developers	161 MB	18,921	The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration...	Windows 32 bit 64 bit
Eclipse IDE for C/C++ Developers	188 MB	6,444	An IDE for C/C++ developers with Mylyn integration.	Windows 32 bit 64 bit
Eclipse for Android Developers	245 MB	3,681	An IDE for developers creating Android applications.	Windows 32 bit 64 bit
Eclipse IDE for Java and DSL Developers	307 MB	2,217	The essential tools for Java and DSL developers, including a Java & Xtend	Windows 32 bit 64 bit

Figure 8: Eclipse Download Page

Unzip the download and run eclipse.exe file to launch the IDE.

Configure test NG. Open Eclipse IDE and go to Help Tab and click on Install new software.

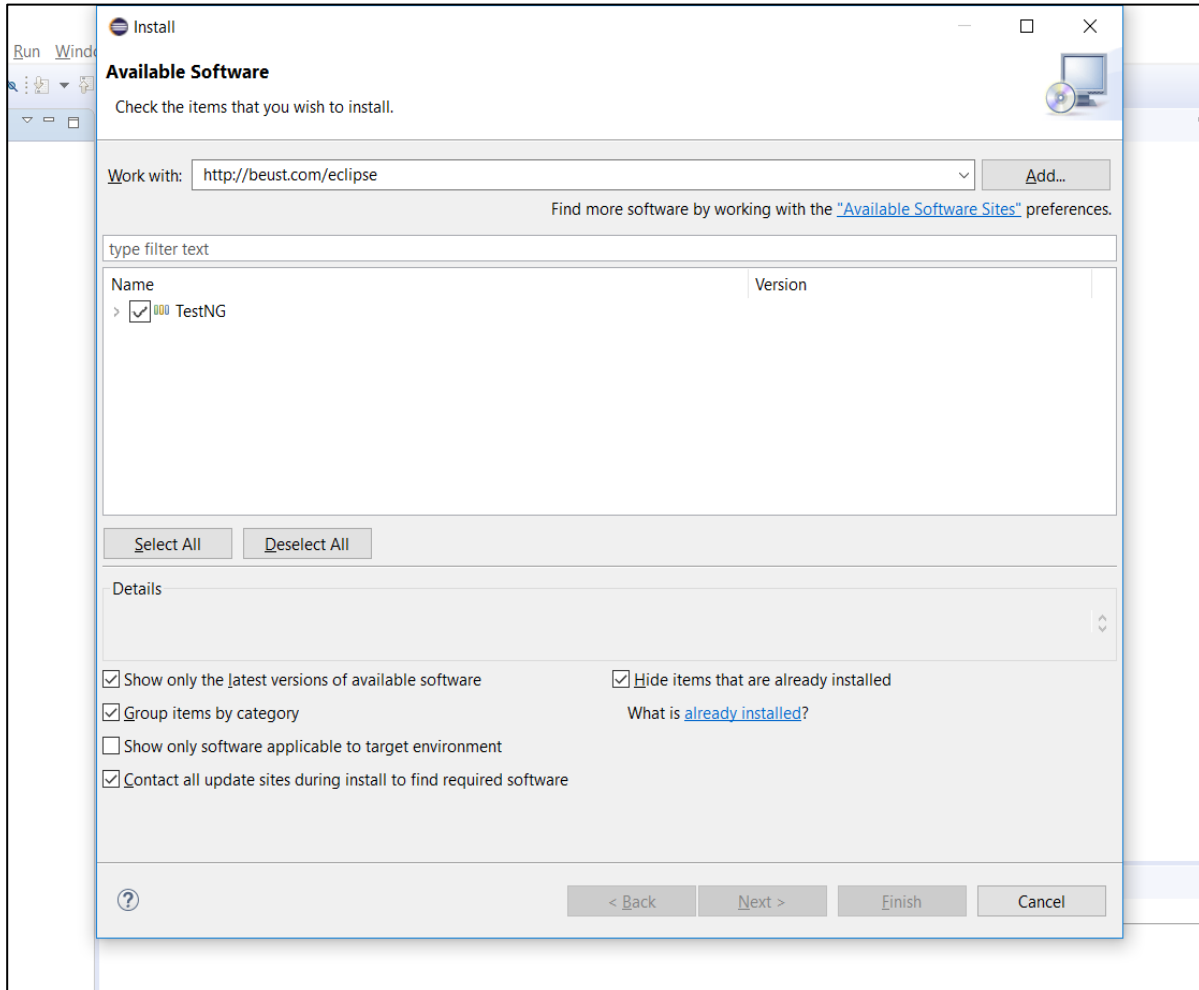


Figure 9: Configuring Eclipse IDE

Enter the URL <http://beust.com/eclipse> and complete the download process.

Configure selenium. Download the Selenium Java Client from <http://docs.seleniumhq.org/download/>

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for [other languages exist](#), these are the core ones that are supported by the main project hosted on google code.

Language	Client Version	Release Date			
Java	3.0.0	2016-10-13	Download	Change log	Javadoc
C#	3.0.0-beta3	2016-09-02	Download	Change log	API docs
Ruby	3.0.0.beta3.1	2016-09-03	Download	Change log	API docs
Python	Selenium 3.0.0.b2	2016-08-03	Download	Change log	API docs
Javascript (Node)	3.0.0-beta-2	2016-08-07	Download	Change log	API docs

Figure 10: Download Selenium Java Client

Downloaded zip file was extracted and saved.

Configure eclipse to work with web driver. Create new java project and then right click on project folder. Clicked on build path and then configure Build path.

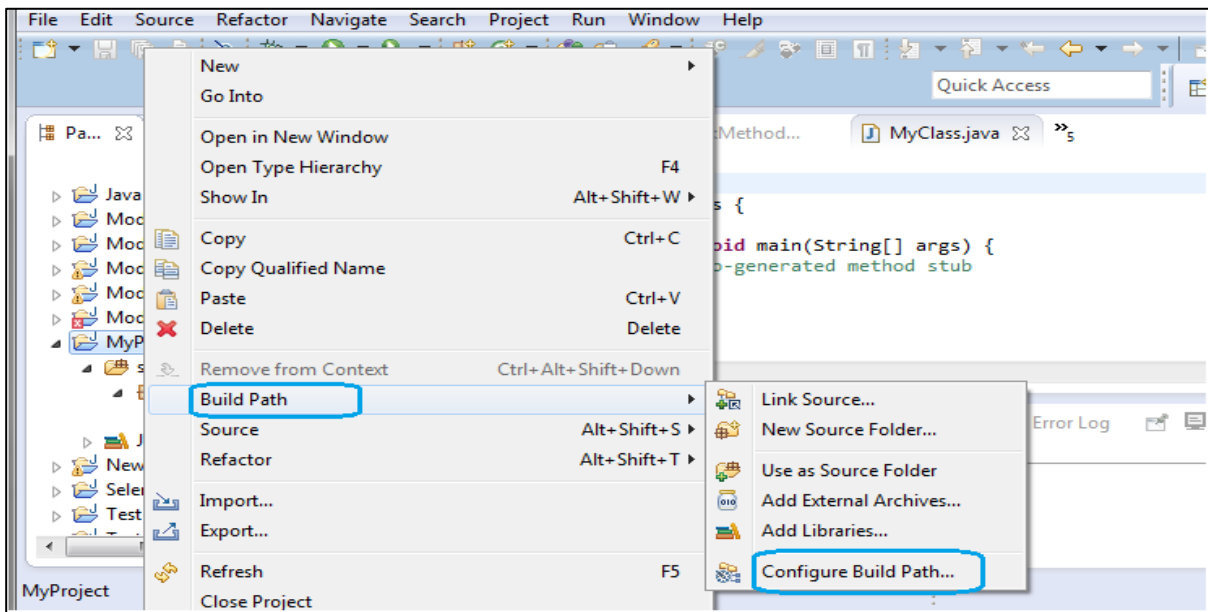


Figure 11: Configuring Build Path

Open the Libraries tab in properties dialog and add External Jars

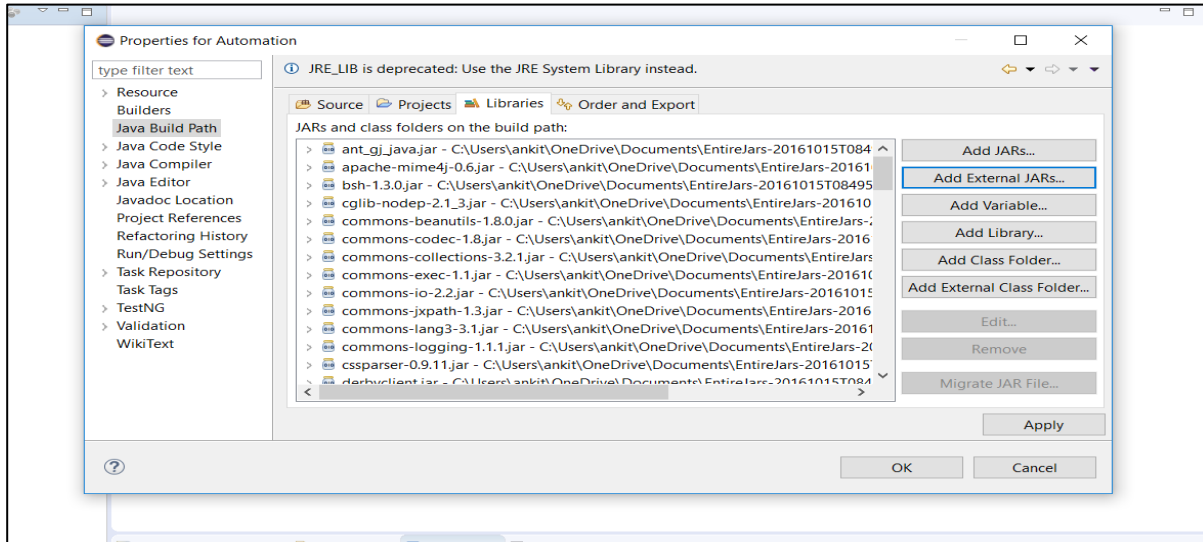


Figure 12: Add External Jars

Add all the jars files that were downloaded in the previous section and then the initial set up is over. After setting up the environment, test script was developed.

IV. Test Script Development

Once the initial environment setup is done, it is the task of testers to develop test scripts. Designing test cases and writing test scripts require lot of skill and thus is the most challenging and time consuming task. Source code for the test will be developed using Eclipse IDE and java is the programming language that will be used to develop all the scripts. Various scenarios or test cases will be created to form test suites. A test case can be added to multiple test suites and test plans. After creating test plan, test suites are created. They are created based on the cycle or based on the scope. It may contain both functional as well as non-functional tests. Table 8 list the sample test cases developed during manual testing and Table 9 is the sample representation of test script.

Table 8: Sample Test Case

Test Case ID	Test Case Description	Test Feature	Test Data	Test Steps	Test Case	Results	Expected Result
F1	Web App UI and Functional Testing when device is not at home	xzgames webapp home screen		Launch xzgames.com page	Verify whether the home page gets displayed		Marketing page should get displayed
		Compatibility		Launch xzgames.com/pair page	Verify the compatibility of the mobile device		If touch is not enabled in the mobile device , then an error message gets displayed after redirecting the user to the home page
		Compatibility		Launch xzgames.com/user/select-profile page	Verify the compatibility of the mobile device		If touch is not enabled in the mobile device , then an error message gets displayed after redirecting the user to the home page
		Compatibility		Launch xzgames.com/user/confirm-profile page	Verify the compatibility of the mobile device		If touch is not enabled in the mobile device , then an error message gets displayed after redirecting the user to the home page
		Compatibility		Launch xzgames.com/pair page with an unsupported device	Verify the compatibility from a laptop/PC		For unsupported devices , an error message has to be displayed after redirecting the user to the home page
		Compatibility		Launch xzgames.com/pair page	Verify the compatibility with a supported mobile device		User should be able to successfully able to navigate to the corresponding pages
		Pairing	Device not at home network	Launch xzgames.com/pair page	Verify that no pairing code will display on screen until press OK		We had some trouble connecting to the set-top box screen should appear containing one edit text box
		Pairing		Enter pairing code	Verify whether there are 7 input text fields in the pairing		Pairing code must be 7 digit number
		Pairing		Launch xzgames.com/pair page	Verify whether a pairing icon is displayed		Before beginning the pairing process , an icon needs to be displayed on the page
		Pairing	Invalid or Wrong code	Launch xzgames.com/pair page and enter an input and click submit	Verify whether the page accepts an invalid pairing code		"Something's not quite right" Screen need to be displayed
		Pairing	Valid Input	Launch xzgames.com/pair page and enter valid input and click submit	Verify that warning message is displayed containing "Continue" button		Warning screen must be displayed with the message " Please connect to your in-
		Pairing		Click Continue Button on Warning Screen	Verify the message ,icon and Continue button on the success screen		Once the pairing is successful , both STB and the webapp displays pairing success
		Pairing		Launch xzgames.com/pair page , enter a valid input, click Submit and then press Continue button on Warning screen and wait for few	Verify that interstitial screen is displayed		Once the pairing is successful STB will keep on displaying thin interstitial screen until game launch

Table 9: Sample Test Script

```

Search Project Run Window Help
Back.java Input.xlsx build.xml SelectProfile.java XZGames.java
25
26 public class XZGames {
27
28     public static WebDriver driver;
29
30     public static TheReporter reportThis = new TheReporter();
31
32     public static String testStatus, testDevice;
33
34     public static String Environment = "";
35     public static String deviceType = "";
36     List<String> dataFromExcel = new ArrayList<String>();
37
38     Logger APPLICATION_LOGS = Logger.getLogger("devpinoyLogger");
39
40     @Test
41     public void f() throws IOException, InterruptedException, AWTException {
42
43
44         DesiredCapabilities cap = new DesiredCapabilities();
45
46         Xls_Reader xls = new Xls_Reader(System.getProperty("user.dir")+"/src/Input.xlsx");
47
48         int rowmos = xls.getRowCount("Devices");
49
50         for(int m=2;m<rowmos;m++){
51
52             String pairingCode = xls.getCellData("Devices", "Pairing Code", m);
53
54             if( !pairingCode.isEmpty() ){
55
56                 String platformName = xls.getCellData("Devices", "platformName", m);
57
58                 String deviceName = xls.getCellData("Devices", "deviceName", m);
59
60                 String platformVersion = xls.getCellData("Devices", "platformVersion", m);
61
62                 String browserName = xls.getCellData("Devices", "browserName", m);
63
64                 Environment = xls.getCellData("Devices", "Environment", m);
65                 String DeviceatHome = xls.getCellData("Devices", "Device_at_home", m);
66
67                 dataFromExcel.add(platformName);
68
69                 dataFromExcel.add(deviceName);
70
71                 dataFromExcel.add(platformVersion);
72
73                 dataFromExcel.add(browserName);
74
75                 String Envur1 = null;
76
77                 switch(Environment){
78
79                     case "0a": Envur1 = "https://0a.xfinitveames.com";
80

```

Data Collection

Data will be collected and analyzed based on the historical observations and automation reports. Data collection process focuses on amount of time difference in execution before and after. Since cost is directly proportional to the amount to time, exact cost difference will be calculated based on the data generated after running complete regression suite.

Summary

The chapter explains in detail about the step by step process involved in the project implementation and data collection methodology. First section talks about the tool selection and later detail plan are discussed here. In the next section, we will focus over data analysis.

Chapter IV: DATA ANALYSIS

Introduction

Data will be collected through the generated reports. After completing the primary humongous task of writing the scripts, this section will show the process of running the automation scripts. This chapter will discuss the report interpretation and conclusion will be drawn.

Data Analysis

Once the automation scripts are developed, regression suite is scheduled to run and generate the appropriate report indicating total number of failed and pass scenarios. It will also mention total time taken to complete the process. Reporting plays very important role in determining ROI (Return on Investment). These reports are shared with the team and clients to track the testing progress and discuss the results.

Selenium is an automation tool and it generates the console output only. To generate interactive reports, we need to integrate selenium with third party tools. The testing framework or build management tools take care of the reports. Test NG and Junit are the two most common types of frameworks. Once we configure our project with the testing framework there is no need of writing an extra code to generate the reports.

In this project, we have used Test NG framework to design automation framework. It is designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems

made of several classes, several packages and even several external frameworks, such as application servers).

Writing a test is typically a three-step process:

- Write the business logic of your test and insert TestNG annotations in your code.
- Add the information about your test (e.g., the class name, the groups you wish to run, etc...) in a testng.xml file or in build.xml.
- Run TestNG [7].

The results of the test run are created in a file called index.html in the directory specified when launching Suite Runner. This file points to various other HTML and text files that contain the result of the entire test run.

Create report: To create report first we need to run the complete automation suite. To run the automation suite, right click and select Run As. Submenu will be open and select as Test NG Test. After clicking, automation will start.

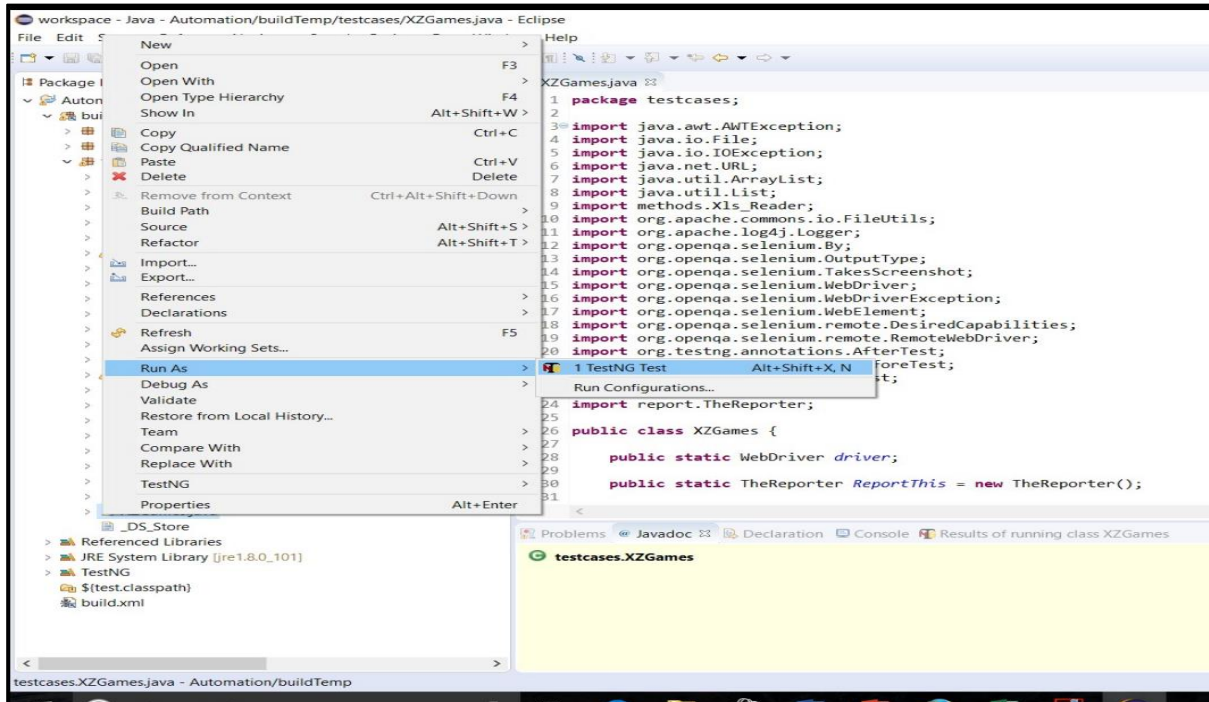


Figure 13: Running TestNG Script

After starting an execution, we will start getting console notifications about the status and wait until the program is getting finished. Once the execution got finished, refresh the project and reports folder will generate automatically.

TestNG report XML, and JUnit report XML files. These files can be found under the output report folder (in this case, test-output)

The generated index.html report contains all the hyperlinks related to automation framework. It contains links to detailed info like test, groups, Ignored Methods etc. Clicking each link will reveal further about the test suite and its components.

The screenshot displays the TestNG Index.html report. The main header is "Test results" with a sub-header "1 suite". Below this, there is a section for "All suites" which includes a "Regression suite" summary. The summary shows "Info" and "Results" sections. The "Info" section lists "testng-customsuite.xml" as the selected suite, along with "1 test", "0 groups", "Times", "Reporter output", "Ignored methods", and "Chronological view". The "Results" section shows "432 method, 384 passed, 36 Failed, 12 Skipped". Below the summary, the XML configuration for the suite is displayed, showing the suite name "Default suite", test name "Default test", and a single class "com.xz.games.sfo.EVCFLOWTest".

Figure 16: Index.html Report

The second report which is most useful for analysis and which is most commonly share among the team is Default Test Report. It gives the complete information about the execution cycle.

The current project start date, time, pass/fail status all are mentioned in this report. It also gives detailed Test step pass/fail status.

Regression test			
Tests passed/Failed/Skipped:	384/36/12		
Started on:	Thu Aug 17 14:31:08 EST 2016		
Total time:	12960 seconds (12960000 ms)		
Included groups:			
Excluded groups:			

(Hover the method name to see the test class name)

Regression Suite			
Regression Test - PASSED			
Test method	Exception	Time (seconds)	Instance
provideModify			
Test class: com.xz.games.sfo.ElineE2EModifyTest			
Hide output Show all outputs			
1 [Pairing page]Validate the Text - Enter the code displayed on your TV Pass			
2 [Pairing page]Validate if the User Image is displayed Pass			
3 [Pairing page]Validate the Text - Ready to pair this device Pass			
4 [Pairing page]Validate if the Submit button is present Pass			
5 [Pairing page]Validate if the footer appears Pass			
6 [Pairing page]Validate if the footer text Pass			
7 [Pairing page]Validate if the TOS works Pass			
8 [Pairing page]Validate if the Privacy & Cookie Policy link works Pass			
9 [InvalidPairingCode]Validate the text Let's try that again. Fail			
10 [InvalidPairingCode]Validate the text Please check your pairing code and enter one more time. (XG-200) Fail			

Figure 17: Default Test Report

This is an email-able form of html report. It contains the same information. In the figure below a snap of the generated report showing the detailed status of failed reports.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Regression suite						
Regression test	384	12	36	12,960,000		
Class				Method	Start	Time (ms)
Regression suite						
Regression test — failed						
com.xz.games.sfo.EVCFLOWTest				verifyCeaseOffLineEVCandOnLineUNI	1472468600519	97016
com.xz.games.sfo.ElineE2EModifyTest				provideReplaceSite	1472468061725	74040
com.xz.games.sfo.ElineE2ESuspendResumeTest				provideSuspendResume	1472468135766	70234
com.xz.games.sfo.ElineE2eTest				ceaseOneEVCTwoUNIRemainsActive	1472468206001	73282
				provideEVCAndUnitConnectedToExistingUNI	1472468403754	47783
				provideEVCUnchangedTwoUNI	1472468451537	117612
com.xz.games.sfo.amend.AmendFullReplaceSiteTest				amendFullChangeReferenceTest	1472465918163	208854
				amendFullReplaceSiteTest	1472466127021	106793
com.xz.games.sfo.amend.E2EAmendCancelTest				cancelOneUNIAfterUSMCreatTest	1472466233815	118491
				cancelOneUNIBeforeTWMCreatTest	1472466352307	104821
				fullCancelAfterUSMCreatTest	1472466457129	108406
				fullCancelBeforeTWMCreatTest	1472466565535	74433
com.xz.games.sfo.amend.E2EAmendDeltaCancelTest				amendFullFullCancelTest	1472466639969	135138
com.xz.games.sfo.amend.E2EProvideCeaseDeltaCancelTest				deltaCancelOnCeaseFullLineTest	1472466775165	164426
com.xz.games.sfo.amend.E2EProvideCeaseFullAmendTest				fullAmendOnCeaseFullLineTest	1472466939645	194319
com.xz.games.sfo.amend.E2EProvideDeltaModifyFullAmendTest				provideDeltaModifyFullAmendTest	1472467352781	49082

Figure 18: Email-able Report

Selenium with Test NG generates basic html or xml reports that lacks rich formatting features. To enhance the report feature XSLT report is helpful. It provides user friendly UI and detailed description of test suite results.

XSLT stands for XML Style-sheet language for transformation, it provides very rich formatting report using TestNG framework.

Following is the pre-requisite to generate XSLT report.

1) ANT build tool should be install (Its necessary to install ANT for XSLT reporting feature). ANT is used to compile the source code and creating the build. It is also very much extensible. Refer this link for steps to download and install ANT.

2) XSLT package downloaded.

3) Selenium script that should be executed by TestNG [8].

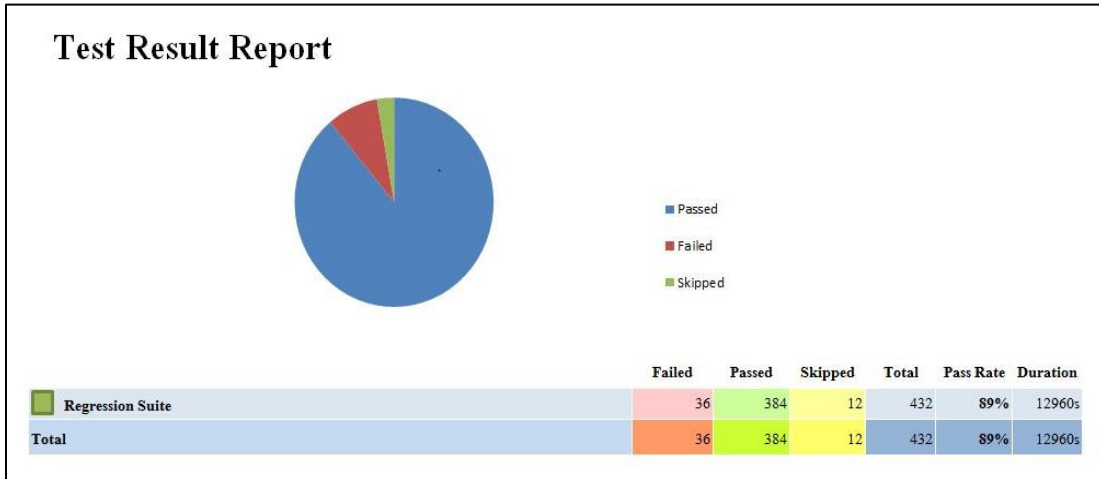


Figure 19: XSLT Report

Data Presentation

Based on the generated reports following comparison study are drawn.

Table 10: Manual vs Automation Comparison

	Manual	Automation
Total No of Test Cases	432	432
Total Time taken for Test Planning	$432 * 11 / 60 = 79.2$ hours (approx. 80 hrs.)	$432 * 70 / 60 = 504$ hours (approx. 500 hrs.)
Cost for Test Planning (assuming Avg. rate \$65/hour)	\$5,200	\$32,500
Total Time taken for Test Execution	$432 * 12 / 60 = 86.4$ Hours (Avg. time for execution 12 Min)	$12690 / 3600 = 3.525$ Hours
No of days for Test Execution for a team of 3 considering 8 hours working day	$86.4 / 24 = 3.6$ (approx. 4 days)	Equivalent to 1 person working for half day
Cost of Test Execution every time (assuming Avg. rate \$65 /hour)	\$5,616	\$0
Cost of running Regression suite in a year (20 product releases)	\$112,320	\$0
Total Cost (Test Planning + Test Execution + Regression in year)	\$123,136	\$32,500

Summary

This chapter explained the report results and its generation methodology. Based on the results, data is analyzed between manual and automation execution. The next chapter explains in detail the results and conclusion drawn from the study.

Chapter V: RESULTS, CONCLUSION, AND RECOMMENDATIONS

Introduction

This chapter provides the results and conclusion of the project. It also provides answers to the project questions discussed in the earlier section. Further recommendations are provided for scope of improvement.

Results

Based on the analysis done in the last section, following results are presented and discussed here.

This following graphs are the representation of the same.

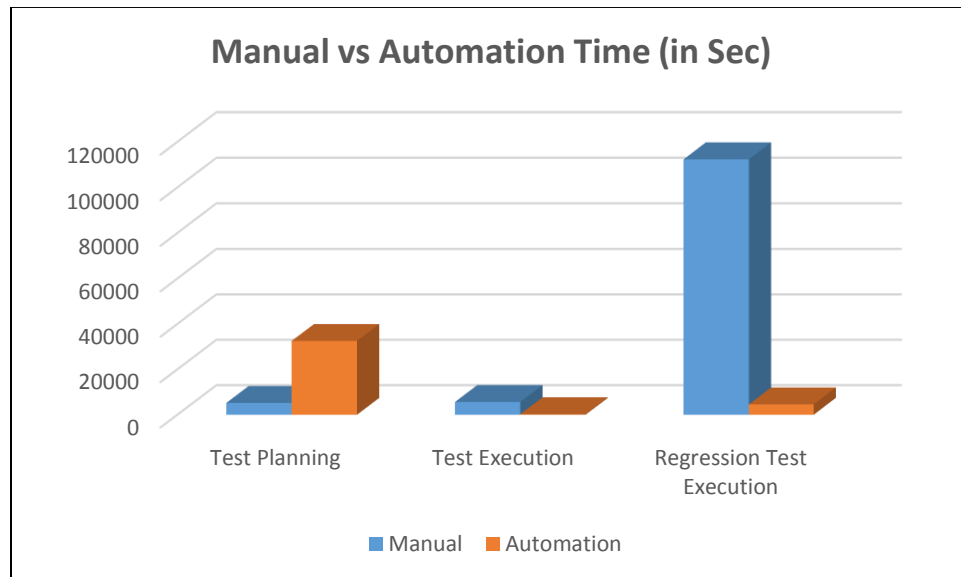


Figure 20: Manual vs Automation Time Study

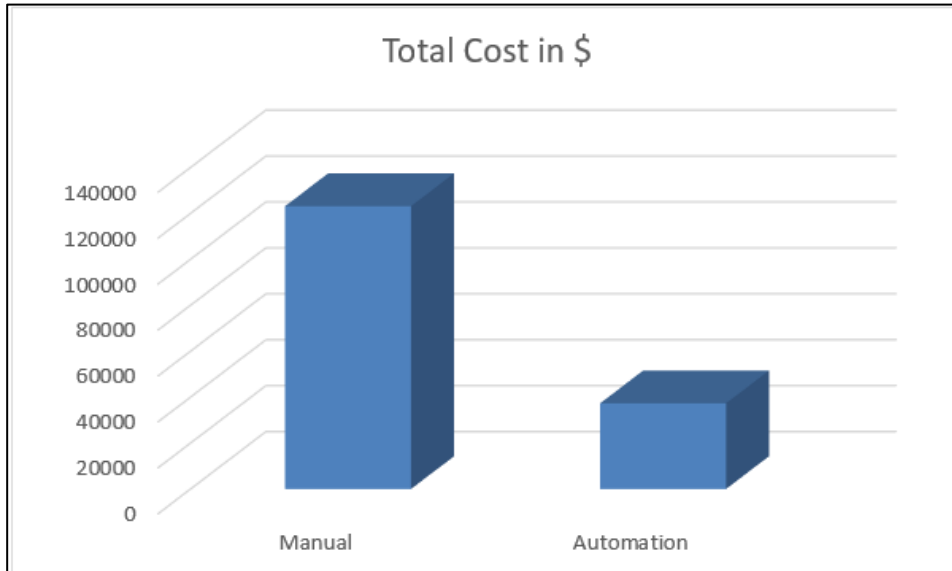


Figure 21: Total Cost of Testing

Based on finding it can be concluded that automating the regression suite is highly successful.

Let's delve into the project question to identify whether the project goal is achieved.

1. How much is the total reduction in turnaround time of completing regression test suite?

After automating regression suite total reduction in turnaround time is more than 82 hours which is more than 24 times of improvement.

2. What are the total savings due to automation?

Based on the results it is evident that test designing and planning for an automation test suite is much higher than manual test planning. But looking the results over a project life of a year, total savings due to automation is \$90636.

3. How the automation testing tool is selected?

Choosing the testing tool was the team effort. Choosing the right testing is the key to success of an automation. Based on the detailed discussion on the topics mentioned in the Tool selection section and separate study conducted by the team, Selenium is selected which proves to be very efficient.

Conclusion

Based on the results it can be concluded that automating test suite is highly successful. Huge reduction in testing turnaround time, cost and software deployment time make the project highly competitive. Quality is optimized as more number of times regression suite can be run which in turn give more confidence over the product.

Project requirements and specification rapidly change to satisfy customers; it is necessary to deliver an error free product on time. Automation the test suite lead to faster execution which in turn optimized the software quality by running more number of tests in lesser time. It also helped in faster deployments of code and thus overall cost is reduced.

Recommendations

- Automation test suites is highly recommendable and provide an edge over manual automation, but balance must be maintained. It is highly recommendable to identify the right test cases for automating. Test planning and design of automation is expensive than manual and the test

cases which run very few number of time over the software life cycle should not be automated, as the overall cost will be much higher than the benefits.

- Success of automation largely depends upon the right testing tool, therefore looking the strengths and weaknesses of your current project and discussion with the team is highly recommendable in choosing the right tool before opting for an automation.

References

- [1] M. Ballou, M. (2008, June). "Improving software quality to drive business agility," June 2008, http://www.coverity.com/library/pdf/IDC_Improving_Software_Quality_June_2008.pdf, August 2016.
- [2] "About us," <https://business.comcast.com/about-us>.
- [3] Wikipedia, "Comcast," <https://en.wikipedia.org/wiki/Comcast>.
- [4] T. Garrett, "Implementing automated software testing—continuously track progress and adjust accordingly," <http://www.methodsandtools.com/archive/archive.php?id=94>.
- [5] D. Hoffman, "Test automation architectures: Planning for test automation," http://www.kaner.com/pdfs/automation/Auto_Arch.pdf.
- [6] J. Silberman, "The advantages of manual vs automation testing," August 2015, <https://www.blazemeter.com/blog/advantages-manual-vs-automated-testing>.
- [7] "TestNG," <http://testng.org/doc/documentation-main.html#introduction>.
- [8] "XSLT report in selenium," <http://www.guru99.com/xslt-report-selenium.html>.