北京交通大學

Beijing Jiaotong University

Bachelor’s Degree Thesis

Thesis Title:

Automation Testing for Improving Software Maintenance and Expansion

School: School of Software Engineering

Major: Software Engineering

Author:

Student No.:

Supervisor：

Date: 2016-05-09

AUTHOR'S DECLARATION

Thesis Proposal for Bachelor Degree Approval Form

Thesis：<u>Automation Testing for Improving Software Maintenance and Expansion</u> School：<u> Beijing Jiaotong University </u>Major：<u>Software Engineering</u>

Problem diagnosis:

I did my internship at Symbio, a software company located in Shangdi, Beijing. I was an automation engineer for the localization testing project; my tasks there were mainly creating test scripts and fixing broken ones. There, I used many testing frameworks, such as Selenium, TestNG, and SeLion. The idea for my thesis came from the project that I did in the intern location.

Enterprise and business software have a very long lifespan of average 10 years with larger software having longer lifespan [1], compared to the average lifespan of consumer applications for iOS of just 30 days [2]. Even though they have significantly different lifespan, both of them go through a typical software lifecycle; design, development, and maintenance, though at a different scale.

Throughout a software's lifespan, it will receive many patches to fix discovered bugs and / or add new features into it. However, no software that is entirely bug-free. Furthermore, the later the bug is discovered, the more expensive it becomes to fix [3]. In the world of enterprise and business, released software have to abide more strictly to software quality standards (e.g. ISO/IEC 9126) compared to typical customer software. Thus, theoretically, regression testing is required before every release to ensure as few bugs as possible get released and thus ensuring quality.

However, doing regression testing manually would require tremendous amount of worktime and potentially delay the release of the software. Moreover, the amount of testing required would increase hugely for every new features and locales introduced to the software, as potential amount of bugs grow with number of line of codes [4].

On the other hand, doing just sanity testing would save a lot of time during testing period prior to software release, however the scope of sanity testing is not enough to ensure the final quality of the software.

As the software grows, eventually more developers are involved in developing and maintaining the software [4], which would produce complexities in communication within, between, and across teams.

Proposed treatment

To achieve high software quality for each release, while maintaining the same amount of time, the best method is to automate testing as much as possible. Automation tests would be able to cover almost all aspect of a software: performance, usability, security, error-handling, etc.

In order to be able to automate the, it would require a custom web app that acts as a central command that executes test cases, store test results, and show the overall testing progress. It would also need to connect with some external services to provide full functionality, such as GitHub. Furthermore, it would also implement test result grouping (based on failure cause, group, priority, type, etc.) and auto-execution functionality to repeat any failed test cases with known environment causes (network, I/O, etc.). For the central command, I would develop it by myself, using existing framework and libraries.

For the test execution itself, SeLion would be used as the main framework; it is a relatively new Java framework that encapsulates both TestNG and Selenium, and able to test website, computer software, as well as mobile application [5]. As it already encapsulates TestNG, test cases can be tagged and grouped for easier management & execution by the central command.

The main benefit of using this method is the considerable amount of work time and resources saved during the testing process of the software, especially in the long term, because the tests can be run at any time. Since it is automated, tests can be repeated as necessary, even in future versions of the software, as long the tested parts haven't been modified. Thus, it would become much easier to ensure the high quality of the software.

Furthermore, as the system is accessed from a web page, it would allow much easier progress tracking and communication between different teams, as well as easier integration of off-site members to join the project.

However, there is also limitation to automation testing. Some functions that require real human interaction, such as fingerprint reader, can't be automated, or any functions that are too costly or complicated to automate.

Before any automating can be done, it would take a lot of effort and resource to setup the necessary environment, such as developing the necessary modules or connecting with 3rd-party services. As such, it is basically another software project; therefore it needs its own documentation, maintenance, managers, etc, which adds further complexity to the software project.

1 REFERENCES

[1] T. Tamai. [Online]. Available: http://tamai-lab.ws.hosei.ac.jp/pub/icsm92.pdf. [Accessed 16 March 2016].

[2] G. Yardley, "Slide Share," pinch media, 18 February 2009. [Online]. Available: http://www.slideshare.net/pinchmedia/iphone-appstore-secrets-pinch-media. [Accessed 16 March 2016].

[3] R. S. Pressman, in *Software Engineering: A Practitioner's Approach*, 7th ed., China Machine Press, 2011, p. 309.

[4] S. McConnell, "Less is More: Jumpstarting Productivity with Small Teams," October 1997. [Online]. Available: http://www.stevemcconnell.com/articles/art06.htm. [Accessed 16 March 2016].

[5] PayPal, "SeLion, Documentation," [Online]. Available: http://paypal.github.io/SeLion/html/documentation.html#what-is-selion. [Accessed 16 March 2016].

Scheduling for deliverables:

No.	Work content	Date	Remark
1	Proposal Document	2016-03-16	

2	Internship Weekly Report	2016-05-02	
3	Internship Final Assessment	2016-05-02	
4	Thesis First Draft	2016-05-11	

Supervisor Approval:

____________________ ____________________

Signature of Supervisor Date

ABSTRACT

Software has been highly pervasive in today's information age in any businesses, be it a tool, or even a service. Such software has long operational lifetime and receive many incremental updates, and have to abide to strict quality control throughout their lifetime. For each update, such software must undergo many quality assurance processes to ensure quality, which would need considerable time and resources, or sacrifice quality to achieve faster deployment. Manual testing is far from ideal since tests will eventually be repeated, and thus there will be a lot of wasted effort spent on the project.

Furthermore, in today's globalized age, people are much more mobile than ever; members of a software project can be located in different room, city, or even different country. Although communication can be easily performed through electronic mail or chat software, members must be able to directly access the latest information of a project from anywhere to perform their tasks quicker and correctly.

In order to test large software product more efficiently for a long period, it is much more efficient to perform automated testing instead of manual testing. Automated test scripts will only need resources during the test case development period and any maintenance needed in the future, which if summed, much less than the total amount of effort for doing the same test manually in the same period and same count of repeated execution. Higher efficiency achieved because test scripts may contain same steps which can be defined in unified methods, and sometimes test cases are differentiated just by different parameters.

However, in addition to automation testing, it will be complemented by a web-based control system which can keep track of test cases and also able to execute test cases, along with other core functionalities. Through this system, a software project can achieve a higher efficiency in maintaining and expanding the software product because of the single interface for controlling test cases and reviewing progress. Better communication and team integration can also be achieved as project members, no matter where they are, can run test cases and view their results from anywhere.

Validation of better efficiency through automation testing with the control system is done through performing a same test manually, by plain automation, and by running that test script through the system. Even though for one-time execution all of them take the same amount of time, automated tests can be performed outside working time and have much lower total effort needed in the long term. Furthermore, the control system will make test case management much more manageable and reviewing overall progress much faster.

Keywords: SeLion, Automation Testing, Long Term, Software Expansion, Software Maintenance

CONTENTS

2 INTRODUCTION

In order to make the objective and methodology of this document much clearer, an introduction has to be made. Introduction consists of background of the problem this document is trying to solve, project scope and description, as well as the organization of the whole document.

Automation testing, or 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 [1]. It can be used to test almost any part of software testing, including performance and GUI[1] testing. Automation testing is critical in software development style with continuous testing and continuous delivery, such as Agile, especially since Agile is used by 84% of the company of correspondent software developers [2].

2.1 BACKGROUND

There exist myriad of software in today's information world, and part of them provide services for consumers, tools for businesses, or even as infrastructures for enterprises, organizations, and other entities. Such software has considerably longer lifetime and will receive many updates to add functionalities, modify existing ones, or to fix discovered flaws.

On average, business software have average lifespan of 10 years; the more lines of code such software have, the longer their lifetime would be [3]. Those ages are much longer compared to the average lifespan of mobile application for consumer, such as on iOS, which is

[1] GUI = Graphical User Interface

only about 30 days [4]. With longer lifespan, that software also has longer maintenance period, which in total will cost a lot. The industry's average for maintenance cost is 80% of the software's total cost [5].

Moreover, software that provide online services (e.g. PayPal) or act as business infrastructures / tools will handle a lot of critical data on daily basis. As such, they have to undergo strict QA[2] processes to ensure data integrity and accuracy. However, as they would receive incremental changes along their long lifetimes, continuous QA processes are needed. One of the most important aspect of any QA is the software testing, in order to discover any bugs.

Performing manual testing is very time and resource consuming, especially if such process is repeated for every changes made into the software. Sanity testing[3] is less time and resource intensive compared to regression testing[4], and it can provide a glimpse into the current quality of the software. However, it doesn't have enough coverage to ensure the final quality of the software. Most of the time, companies sacrifice quality to achieve deadline, or vice-versa.

Manual testing also wastes a lot of resources in the long run; often, test cases contain same steps which if they were a script, parts of them could just be copied. Even more, some test cases are just a matter of different parameters. These minor differences mean that manually done tests have to cover all of them each time, which in turn considerably increase the total amount of resources spent on testing during the entire lifetime of the software.

Besides the matter of testing itself, management plays an important role in a project. High percentage of executives, as high as 80%, believe putting importance on management helped them during the recession [6]. Thus, the system must have some form of report

[2] QA = Quality Assurance.
[3] Sanity testing = Smoke testing; tests to discover feasibility and possibility of further testing of a software.
[4] Regression testing = testing that includes all part of a software.

mechanics, especially in diagrams or charts, since they are the easily understandable.

Lastly, as information and transportation have progressed a lot, members of a software project can be working from different places, or even different countries. Even though communication can be performed easily and cheaply using various communication methods, members must be able to access the data about the project directly and easily to ensure optimal work performance and avoiding misinformation regarding the project.

2.2 PURPOSE

The purpose of this document is to provide overview, requirement analysis, software design, and expected outcome of the web-based automation testing control system, which would complement automation testing used in software. Furthermore, this document contains discussion regarding usage of automation testing, as well as some data taken from simulation of a testing done on a software product.

However, it must be noted that the system described in this document is just a proof-of-concept; it only contains core functionalities with limited modifiability, and using more approachable components rather than the most advanced / industry’s latest standards software components & technologies. Furthermore, all of the necessary components for the entire system to work will be deployed in a single machine, in contrast to the ideal setup of independent hardware for each component: system server, database, and execution environment.

The design is meant to show the potential benefits if it were designed and deployed in a software development environment.

2.3 SCOPE

This project consists of the design and implementation of the prototype of the automation testing control system. The automation testing control system will be entirely web-based, hosted in a local machine, that have the ability to connect to database for data recording / reading, as well as processing test results. The system will be designed to work specifically with SeLion.

The system serves as an interface between users and test execution environment, as well as a control center for the registered test cases and their results. Additionally, the system has connection to a network / online VCS[5], such as GitHub for this project, to download the latest test scripts, as inputted in the execution parameters.

2.4 PROJECT DESCRIPTION

The main objective of this system complementing typical automation testing is too make software maintenance and expansion more efficient in the long term. It is done through choice of technologies, clever software design, and some experience in the world of automation testing.

2.4.1 Perspective

The automation testing control system is meant to be a centralized command center to execute, manage, and review test cases. The system can manage test cases for many platforms, as long as they are supported by the main testing framework, SeLion.

[5] VCS = Version Control System

The core framework of this project, SeLion, is a Java-based, open-source framework which encapsulates several popular testing frameworks, such as TestNG and Selenium, with additional features on top. Compared to bare Selenium, SeLion can test more types of software (desktop, mobile, and web applications) and make it easier to write and maintain test cases. However, since it contains a lot of additional feature, SeLion is generally slower than Selenium.

This system is designed to be easily developable by a company's own development team, and thus the system will be written in Java and uses frameworks and/or libraries written for Java. Furthermore, in order to limit the client application to a single version, the system uses web-based application as its user interface so that it is

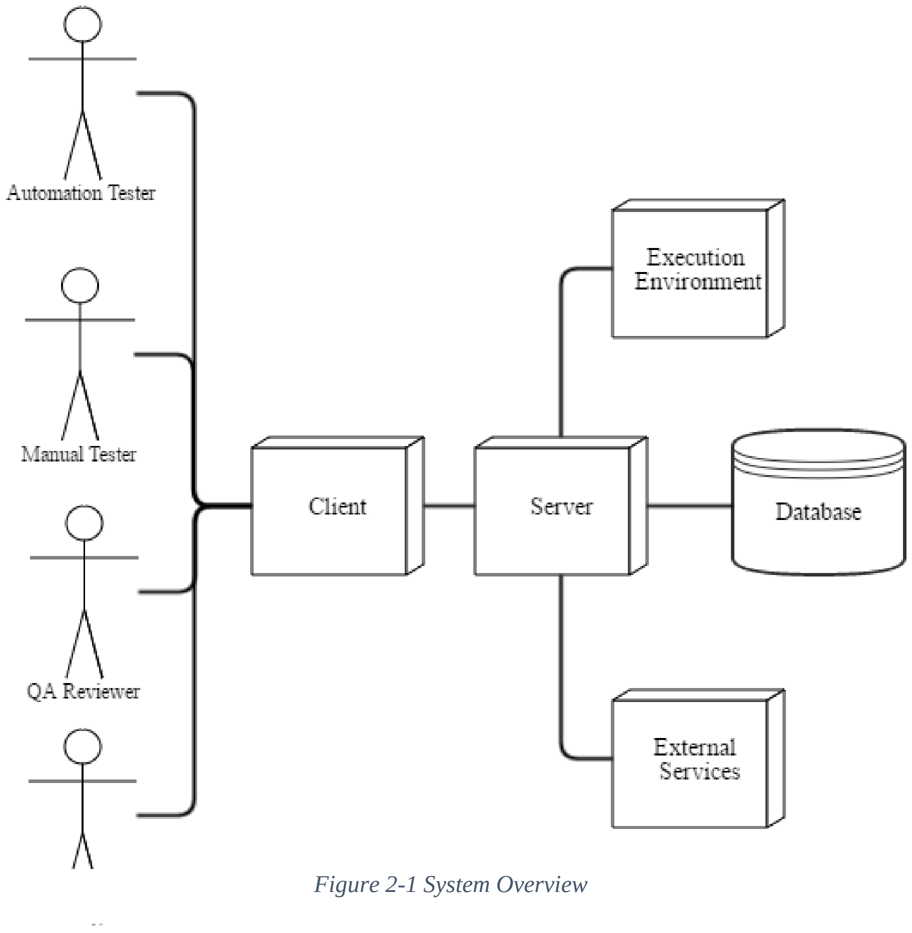

Figure 2-1 System Overview

accessible from any platform, as long as it can support an internet browser.

Moreover, the system is also designed to improve team collaborative effort in testing through the use of a singular location for all team members to do their testing tasks. Members must be able to add notes on stored test case results, so that others can learn from others’ mistakes, and thus improve the overall productivity of the team.

There are various commercial alternatives that have similar core functionalities than this system, and some of them offer more features. However, most of them require usage fee which increases the software development cost, and offered as a desktop application which limits usability of the software.

Therefore, the system must use easy to develop technologies to enable in-house development of the system, as well as allowing the development team to implement needed changes to the system immediately. Compared to purchasing an automation testing control system, developing one in-house will reduce total expense in the long run, albeit requiring extra effort in the beginning.

The overview of the system is pictured in Figure 1-1; users will use the system through the same client, which is the client side web application. From there, the client will communicate with the server which is connected to other components.

Execution environment is the one that executes the test scripts; it is a server version of SeLion containing the SeLion grid. The grid controls nodes (machines that do the actual script execution) by sending cases to them, and then passes the results back to the client, in this case, the automation testing control system.

2.4.2 Functions

The control system has several core functionalities:

- Test case execution
- Adding test case result metadata
- Viewing running test cases
- Viewing test results with filtering & grouping
- Automatic failure grouping and re-execution

2.4.3 User Descriptions and Characterizations

The system has several types of key users:

- **Automation Tester:** Automation engineers are tasked to writing test scripts, executing them in the system, and analyzing any failed cases. After they have written their test cases, they would first test those scripts locally in their machine. Then, they would upload them to a VCS, and execute them using the system with customized parameters as required by the test case. After those test cases have been finished, the automation testers would analyze any failed cases and fix them, re-upload them, and execute them in the system several times until achieving successful execution. If some of the cases cannot be passed, automation testers would discuss with manual testers for possible bug(s) which block the case.
- **Manual Tester:** Manual testers main task is to perform manual test cases, and add those test cases into the system. Manual test cases are test cases that cannot be automated, such as biometric authentication system, or cases that are too complicated / requiring special configuration(s) which are not supported by the execution environment.

- **Project Manager:** Project managers will only see the reports of all test cases; they will observe the progress of the testing from the numbers of cases that have been executed, as well as the passing percentage.
- **QA Reviewer:** QA reviewers would review any passed automation test cases with the case designs. They compare steps defined in the design with the steps recorded in the test result, and notify any discrepancies to automation engineers in charge and / or the project manager for immediate fix. They are also tasked with checking the test case results for any false positive result.

2.4.4 Operating Environment

The system will be developed based on Java, and almost the entirety of the system will be developed using open-source frameworks / libraries for Java. As such, the system is very flexible in terms of the operating environment, as long as it can support the following technologies:

- **Java:** A general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible [7]. Furthermore, as Java runs on top of JVM[6], once a Java code is compiled, it can be run on any platforms that support Java [8]. Java is chosen because it is an open-sourced, making it completely free to use, and the most used general purpose programming language on the world, which resulted in various tutorials and plugins available. As a result, the system will be easily expandable through

[6] JVM = Java Virtual Machine.

many 3rd party plugins while keeping the development cost & time as low as possible.

- **Java EE (Java Enterprise Edition):** Java EE platform is built on top of the Java SE platform. The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications [9]. Some of the APIs, such as Java Servlet, is necessary in developing the control system.
- **MySQL Community Edition:** MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation [10]**.** As a lot of developer use MySQL, it is easy to find additional plugins and / or supports should they be necessary. Community Edition is chosen because it is free to use, and has the necessary functionalities to be used in the system.
- **Apache Tomcat:** It is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies [11]. Tomcat will be the HTTP server for the automation testing control system web application.

For the hardware specification, the hardware chosen depends on the volume of work that the system will do as Linux can be run on any hardware configuration. Preferably the system should be run on server-grade machine to ensure reliability and security.

2.4.5 Design and Implementation Constraints

As the system is built around the SeLion framework, projects must have a certain structure to be able to be read and processed by the system. Although SeLion can perform automated testing on almost all form of applications (desktop, mobile, and web), sometimes a project needs another testing framework to fulfill the team's requirements, such as team's inaptitude in developing in Java, or usage of the company's private / proprietary testing framework. The automation testing control system downloads latest scripts from VCS (GitHub), and the project must be in a certain structure to be able to be read and processed. Thus, the system must undergo a considerable modification to be able to process test scripts which are written for other automation testing framework other than SeLion.

Second, even though the system is written in Java, and thus there is huge availability for APIs written in Java to connect with a number of 3rd party applications and / or services. However, the automation testing control system doesn't support web services, and it may become a hurdle when it becomes needed to connect with existing business services in the future.

Finally, the whole system can't be too complicated and / or have too many functions; the system is meant to be a tool to help testing software in order to achieve reduction in time and resource required for any testing. When the system becomes too complicated, the team and / or the company will eventually need to redirect non-trivial amount of resources to maintain the system, which defeats the system's initial goal.

2.5 DOCUMENT ORGANIZATION

This document is broken down into several main chapters, and they are:

- **Introduction.**
  Introduction to the background of this project, its purpose, and description of the project.
- **Requirement Analysis.**
  Based on the problem and goal described in introduction, a set of requirement is made for the software project.

- **Software Design.**
  Architectural design, database design, data flow, and as well as designs in different models.
- **Implementation.**
  Logic designs of core functionalities, as well as their reasoning.
- **Test & Validation.**
  Contains method and result of tests done on this project to show that initial goal has been achieved.
- **Conclusion.**
  Summary of the project and its analysis.
- **References.**
- **Appendix.**
  Contains attachments important to this document, but too large to be inserted within the content.

3 REQUIREMENT ANALYSIS

From the description of the project, the automation testing control system has to have several core functionalities, as well as some non-functional properties to fulfill its objective in improving software maintenance and software expansion.

3.1 FUNCTIONAL REQUIREMENTS

As described in previous chapter, the system must have some core features in order to be able to achieve its intended goals. Such functions are an absolute must to be included in the final product, and must be performing flawlessly.

3.1.1 Test case execution

This feature allows automation testers to execute test cases through the system, by inputting parameters beforehand. The parameters are class name containing the test case name, test case file location, URL[7] to the repository containing the test script and its branch name, and any test case filters to use. Then, the system will queue all test case executions until an execution machine is free.

Test case file location needs to be specified in order to save time from having the system needing to read all files for that specific test case, as well as preventing cases with same names to be wrongfully executed, especially in large testing project.

[7] URL = Uniform Resource Locator.

While waiting for some cases to be finished executed, automation testers can start working on different ones in different branches. Having such customization on the URL to the test script location allow automation testers to work on multiple test cases simultaneously. Furthermore, users can also help others in running their cases just by using the URL to their own repositories.

Filters allow testers to run specific test cases at a time; this is absolutely important, especially when number of test cases have grown large, as well as their parameter variations. Without it, testers need to separate and possibly rewrite each variation of a test case to different files, which make managing test cases more difficult.

Since more than one person will be using the system and the limited number of available execution environment, the system must be able to handle multiple test case execution commands at once. Thus, there must be a queueing system where execution jobs are queued with FIFO[8] method, and processed one by one per available execution environment.

Finally, when the execution has finished, the system will receive the result and then automatically analyze it and saves the result into the database for further viewing by users.

3.1.2 Adding test case result metadata

After finish executing test scripts, the execution environment sends their results to the automation testing control system to be processed and then stored in the connected database. Then, users will be able to see those results through the client, and then able to add some additional information regarding those results, especially failed ones.

[8] FIFO = First In First Out. The first object queued will be the first to be processed.

Some information must be able to be added, such as bug ID for the cause of failed test case, or comment for miscellaneous information. These allow users to directly add and view such information to the corresponding test case within the same page, without having to move to other application, and thus saving some time. Furthermore, these can also improve team productivity, as other members can view those comments and bug ID and may improve their own cases if there are some similarities in the case structure.

3.1.3 Viewing running test cases

Users must be able to view the list of test cases being run and test cases that are in a queue. The list contains test case name, variation (if applicable), added time, current status, and executed time (if already executed). This function is necessary to prevent automation tester running the same test script as another automation tester, and thus prevent wasting time.

3.1.4 Viewing test results with filtering & grouping

Users of the system must also be able to view all of the test case results; if there are multiple version of the same test case, the system will only show the latest successful one. Viewing results is important to know for certain the quality of the work that has been done on the testing project, as well as enabling test case review.

However, if there are large numbers of test cases and group names used in all test cases, it will be very hard to find a specific test case. Thus, the system has built in filtering system that reads all available group names from the database and then lists them as options in the view test result page. Users then choose which groups do they want to view, and then the system will show the filtered list of test results.

Furthermore, based on current list of test results shown on the page, the system also shows current number of test cases, number of passed and failed ones, as well as the passed percentage. This is useful for managers to get a quick glance of the current progress of the current testing period.

3.1.5 Automatic failure grouping and re-execution

In the test result view page, there is an option to group failed test results. Here, the system will automatically group failed test results based on the cause of the failures, and cause of the failure will be taken from the stack trace of the test case. There is a possibility that some test cases have same flaws and a single fix can fix many test cases at once, saving a lot of time. Thus this option is made to make it easier for testers to compare cases with similar failure reason.

Furthermore, users can tag failure reasons for automatic re-execution. With this, the moment another test case has the same stack trace as tagged ones, the system will automatically re-execute it up to twice (configurable) until the test case passes. Some test cases can have false negative due to environment causes, such as unstable network or using a simulated hardware. This feature will lessen the overall time testers need to analyze their failed test cases because the ones with false negatives are more likely already passed by the system, instead by the users.

3.2 NON-FUNCTIONAL REQUIREMENTS

Besides the core functionalities, the system has to meet several important characteristics to complement those functionalities, and ultimately deliver what the system is designed for. These attributes are derived from common standards of software, human’s habits, and also from the design of this system as well.

Lastly, the final result of the system must have as few bugs as possible, since defects found during operation will cost much more than during design phase [12].

3.2.1 Reliability

Software reliability: “The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered” [13].

Reliability is the most important non-functional requirement of this system. As the system track test cases and their results, such data can be highly critical to the team’s progress, especially near deadlines. The team can’t afford for the system to be inconsistent. Therefore, the system must be reliable and available at all times.

- **Data Integrity.**
  Data integrity means how well the system maintain the integrity of any data it handles; in other words, how well the system prevents any of its data from getting corrupted or lost in any situation. Testing is an important part of any software project, no matter its development style (e.g. Agile, Test-Driven). Thus, any testing data are crucial to the team.
  Therefore, the system must have a confidence that any progress made with the system will not be lost nor be corrupted under any situation, so that no additional resources are spent in redoing completed tasks.
- **Quick Recovery.**
  Whenever the system crashes, it must be able to quickly recover and resume its normal operation. When this system is

operational, the team will keep track of all test cases and run all of its automated test cases from this system. As such, the system must be available at all times and have short MTR
If the system takes a long time to recover, the team will waste time and manpower as members will be unable to see testing progress; this is crucial during test scripts debugging period where automation testing engineers are fixing failed test cases.

3.2.2 Performance

Having a high performance means that a software performs certain tasks with high efficiency. Although there is no absolute number for measuring all aspect of a software's performance, some of aspects of performance are judged by users' opinions. Other aspects of performance are measured by calculations, though there is no absolute margin for high performing software and low performing ones.

- **Load Time**
  Having a low loading time simply means that accessing any pages, elements, or data will only take short amount of time. However, the margin for having a low loading time differs depending on the object being load; for example, a web page is considered to have low loading time if it loads under 2 seconds without considering connection speed [14].
  Low loading time is essential especially for a system that will be used often on daily basis. When loading an e-commerce website, anything above 2 seconds is considered unacceptable by almost half of users [14]. Thus, same can almost be said for software developers; high loading time can annoy or even anger users, especially since there is no other option except waiting for the page to be loaded. This attribute is meant for lessening sources of annoyances from members of software project team in order to keep their productivity high at all time.

- **Latency**
  A low latency software is a very responsive software, which means that the software immediately begins executing a command once it is given by the user. Having a low latency doesn’t mean that any command will be completed in lower time, it only means that there is almost no delay between user input and command execution. If it takes a long time to process, usually a software has a graphical method to indicate that user’s command is being processed (e.g. loading bar).
  In a system used by multiple users with all the users depending on any data saved, all command must be executed immediately to prevent repetition of the same task. Task repetition can happen while the system is processing a command sent by a user, a second user sends a command for the same task.
  Furthermore, as stated in previous point, people are impatient. As the client of this system is a web application, users would likely to refresh the page once they feel that the page has gotten unresponsive, while actually the software is still processing their sent commands. This may result the same command sent multiple times into the system, which slows the whole progress of test executions, and ultimately wastes time.

3.2.3 Maintainability

Maintainability describes how easy the effort needed to maintain a software; most of the time, it is measured by the amount of time and resources in total needed to maintain a software, compared to its development cost.

The automation testing control system is designed for improving efficiency of software testing in the long term by cutting the amount of time and resources needed in testing. Therefore, the system itself must also be cost and time efficient to maintain, and such the client for the system is developed as a web application, so that it is accessible from almost any platform.

3.2.4 Modifiability

Modifiability means how easy it is to modify existing features of a software, and also when adding new features. This attribute is especially important for software with planned long lifespan, or software used in highly unstable environment where requirements often change.

In the competitive business world, development teams have to adapt quickly to any new tools or technologies that might appear in the future. For the team to stay competitive, tools they use must also be up-to-date. The automation control system must be readily accepting additional APIs and be effortless to modify so that any changes made by the team can also be implemented quickly into the system, checking unneeded wastes to a minimum.

3.2.5 Usability

Most of the time, usability refers to the amount of effort required by a user to understand and use the GUI[9] of a software. It is affected by words choice, element designs, flow of interface, and method of input for data. According to ISO 9241, definition of usability is: “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use” [15].

Members of a software development team may change often, and the team can’t allocate much time to introducing new members to the system and teaching them how to use it. Furthermore, team members come from different backgrounds and have different career track. Thus, the GUI of the automation testing control system must be easy to use from the first time.

[9] GUI = Graphical User Interface.

3.3 DATA STORAGE REQUIREMENTS

All data will be stored in basic relational database in form of SQL databases, using MySQL. Although the database can be stored inside the same machine as the automation testing control system, it is much preferred to deploy MySQL database in different machine to ensure higher reliability. Moreover, the machine hosting MySQL database should have a form of data redundancy (e.g. RAID[10]) or backup, be it within the same machine or the same network, for guaranteeing data integrity in an event of catastrophic hardware failure since there is no reliable measurement on how close a drive is to failure, as official measurements cannot be trusted [16].

3.4 USE CASES

Based on the type of users described in previous chapter and the core functionalities described in this chapter, there are several basic use cases created from them, as shown in Figure 2-1. Each user type uses different features based on their interests. Their descriptions and explanations are contained within section **2.1,** the Functional Requirement.

[10] RAID = Redundant Array of Independent Disks

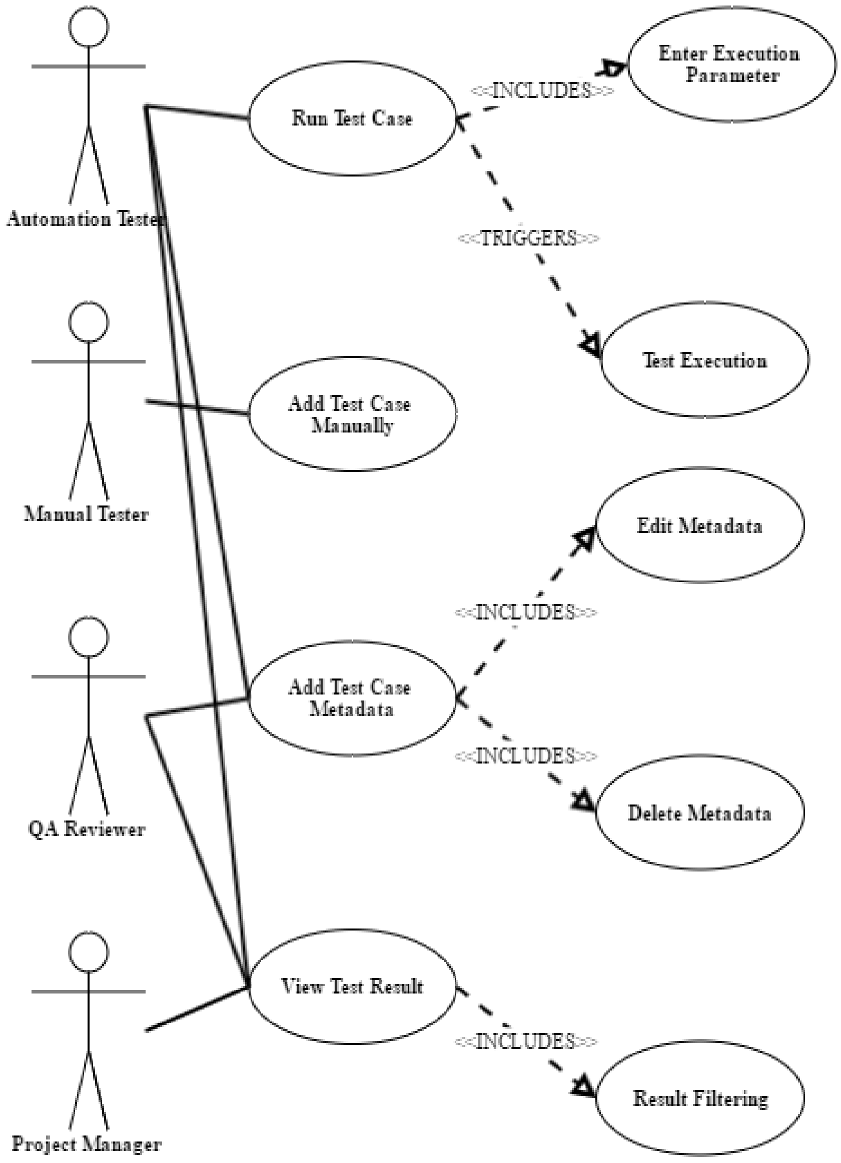

Figure 3-1 Overall Use Case Diagram

3.5 DATA FLOW

Figure 2-2 shows how different data objects moves between elements of the whole system, and who is processing those data. This diagram gives an overall look on how the data moves around the whole system and how it operates during a normal operation.

When an automation tester initiates a test script execution, the client will send the parameters to the control system to be processed. First, the system will queue the task and wait for a vacant slot in the execution machine. When there's a free slot, it will download the corresponding script from the VCS and send it to test execution machine via the SeLion grid to be executed. When it is finished, the execution machine will send back the result into the system, where it will be processed and stored in the database.

The same will also happen during an automatic re-execution of a failed test case, though it doesn't need any input from a user. Older failed test result of the same test case will be immediately replaced by newer ones, no matter if they are passed or still failed.

Next, when a user wants to view test result, the system queries it from the database, and then processes all queried test results before sending them to the client. The processing changes depending on the options chosen by the user before querying, such as enabling failure grouping or choosing filters for test cases.

Finally, in the view test result page, users can add, edit, or delete metadata of any test cases shown in the page; the metadata includes bug ID and a text note. Bug ID is used when there is known and documented causes(s) for the failure; user can refer to the bug documentation services that the team uses. Note is used to store any message that the entire team should know regarding the test case, such as when the failure is caused by problems not yet documented (new bugs), or to let other user know important matters regarding the test case especially when the test case is transferred to other user.

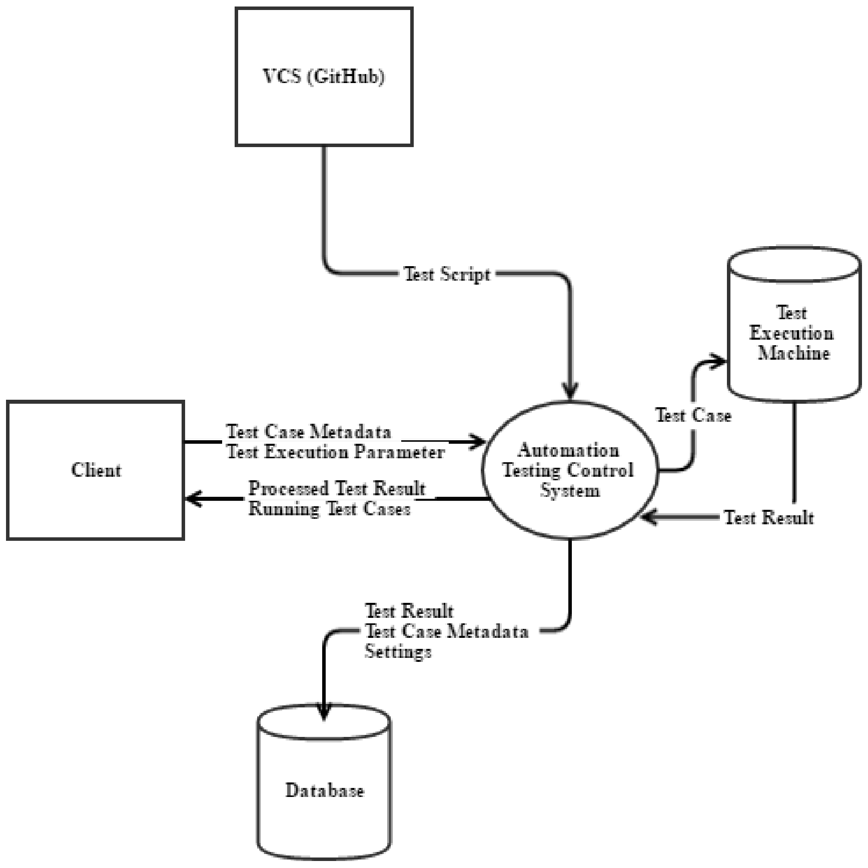

Figure 3-2 Flow of data objects in the system

3.6 REQUIREMENT CONSTRAINTS

There are some constraints that limit certain aspects of the system that cannot or difficult to be overcome. However, those limitations are not critical, meaning that they don’t disrupt any of the core functionalities though they do bring negative impacts if the system is used in real software development environment with real developers.

This system is meant to be developed and managed by a development team, and thus the system must be able to be developed as easily and quickly as possible. The system also needs to have high maintainability to keep the team’s extra expense to a minimum

As such, there are some secondary functions that can be implemented which would complement the primary functions of the system, but omitted to keep the whole system as lean as possible.

For example, the system can implement an authorization and privilege system for all the users. Users will need to login first and are only allowed to use some of the functions while higher ranking developers and administrators can use all of the functions.

Such function is important to prevent any unauthorized access from within or outside of the company, and also restricting users from viewing other data besides those that are allowed to them, thus safeguarding assets. Limiting functions helps protecting data from being wrongly modified or even deleted by new developers, which can be very costly to the team.

4 SOFTWARE DESIGN

Designing a software requires fully described functional and non-functional requirements, as well as the objective and user base of the software. The design of the automation testing control systems is separated into several parts: the design of the system itself, the user interface, and then the schema for the database used by the system.

4.1 ARCHITECTURAL DESIGN

The architectural design of the whole system uses two different architectural style; client – server style on the macro scale, and then the layered architectural style for the design of the control system itself.

4.1.1 Client – Server Architectural Style

The system uses a client – server architectural style, with thin client model. Here, users access the system using the client, and within the server, there is a clear separation between the presentation logic[11] and the business logic[12]. The structure is shown in Figure 3-1.

The whole system consists of several key elements, and they are:

- **User Interface:** The web application where users interact with the automation testing control system.

[11] Presentation logic = set of functions which interpret data into user-readable format, and vice versa
[12] Business logic = set of functions which perform the core functionalities of a software.

- **System Server:** The automation testing control system main server. Controls elements in the entire system, as well as any data passing between them.
- **Execution Environment:** Location where test cases get executed.
- **System Database:** Database used by the system to store test results, as well as any relevant data.
- **VCS:** Repository that contains test scripts.

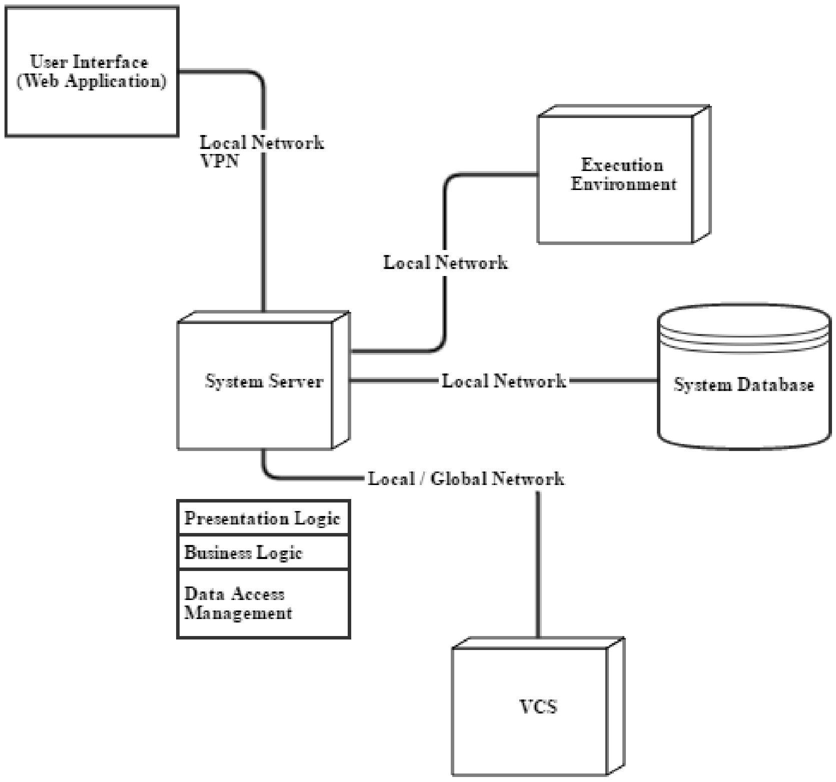

Figure 4-1 Client - Server Architectural Style

As all the logics are contained within the system server, there will be less data need to be sent back and forth between the client and the server, reducing total network load and thus improving responsiveness. Moreover, it also means that less data need to be loaded before users can use the interface, which means faster load time. Additionally, since client has to get any data from the server before being used, users will always get the latest data to work on.

Furthermore, because the entire system is located in a single entity, maintaining and troubleshooting any problems will be much more straightforward and therefore more efficient. Besides that, by keeping any data from being permanently saved in client side, any troubles encountered in the client will not affect any crucial data in any way, improving the reliability of the system.

Having data and functionalities centralized in the server removes any restraints on the location of the client, enabling the system to be accessed from anywhere assuming the necessary network connection is available.

4.1.2 Layered Architectural Style

To achieve higher maintainability and modifiability, the system must be highly modular, or in other words, have good separation between modules. Thus, the internal structure of the system uses layered architectural style. This style limits coupling between layers by separating them with interfaces. On the other hand, it also results in better cohesion within a same layer, which ultimately improves modularity. The overall internal design of the system is shown in Figure 3-2.

The internal structure is separated into 3 main layers: presentation logic, business logic, and data access logic. Presentation logic layer handles all data conversion between user-readable format and the data structure used inside the system, vice-versa. Business logic performs all the core functionalities of the system as well as their dependencies, and each sub-function is made into its own module for better modularity. Lastly, data access logic handles all data operation to external sources; this includes the system database, VCS through API, and finally the execution environment.

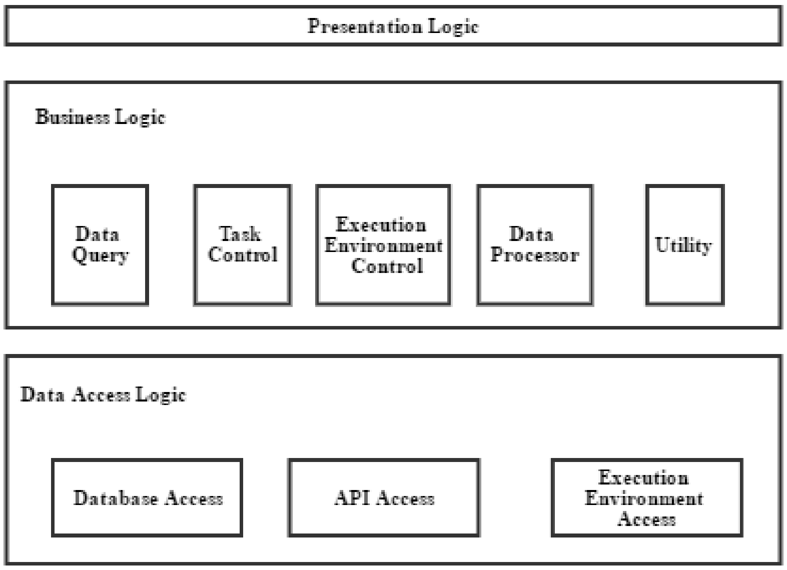

Figure 4-2 Layered Architectural Style

Business logic contains several submodules, and they are:

- **Data Query:** Performs any data request to any connected sources (e.g. database, VCS).
- **Task Control:** Handles queueing of tasks and tracking running tasks.
- **Execution Environment Control:** Control the execution environment, by sending commands and listening to success or failure signals from running test cases.
- **Data Processor:** Process and analyze any data in the system, such as test results for categorizing and grouping failure.

- **Utility:** Contains loosely cohesive functions that are used multiple times across different modules.

4.2 GUI DESIGN

The client side of the system acts as an interface between the user and the whole system. Client has several main pages; each serves as an interface to main functions of the system. All pages will have the same header that contains links to each page that performs one of the main functions.

4.2.1 Test Execution Page

This is the page where automation tester can execute a test case. It contains several text fields for entering several key information for running a test case, as well as a button to execute it. The design of this page is shown in Figure 3-3.

All of the text fields are mandatory, which means all of them must be filled in before user can execute a test case. All text fields only accept ASCII encoding. The text fields in this page are:

- **Test Case Name:** Name of the test case to be tested (e.g. LoginModuleMainTest).
- **Test Case File Location:** File name of the .java file that contains the test case method named in previous text field (e.g. LoginModuleMainTest.java).
- **Repository URL:** URL to the repository that contains the test case file.
- **Repository Branch:** Branch name of the used repository.
- **Test Case Filter:** If a test case have several variations, this allow user to specifically run specified versions.

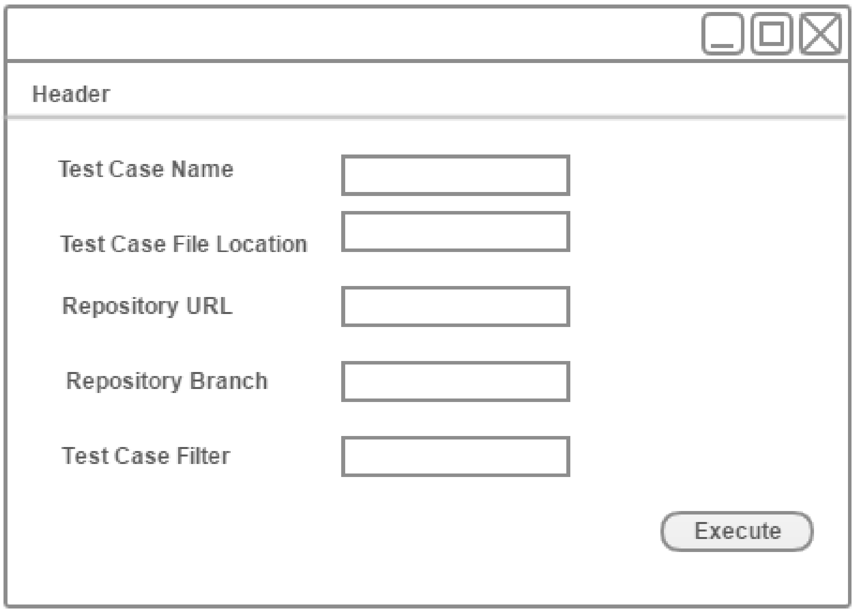

Figure 4-3 Test Execution Page

4.2.2 Test Result View Page

This page lists all of the test results stored in the system, and it consists of several main elements. First, there is a button to enable grouped view of the failed test result. Second, there is a small box on the left side of the window, which lists all available groups based on groups attached to available test cases.

Third, the "View Metadata" button allows user to view, add, edit, and delete further details of a test result in another page. User need to pick a test result first by clicking on that row, and the clicking on the "View Metadata" button. Lastly, the main table view lists all available test results with some important columns. The design is described in Figure 3-3 and Figure 3-4.

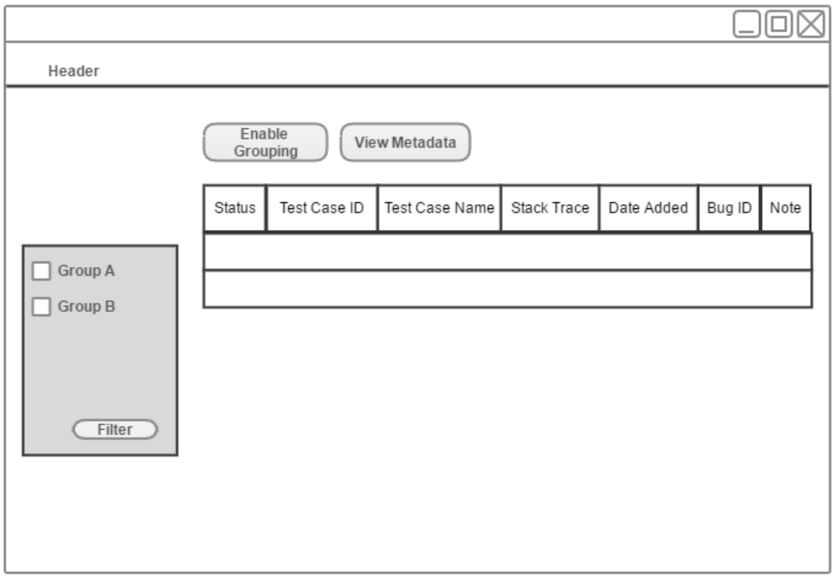

Figure 4-4 Test Result View Page

When the user enabled grouped view, the column names slightly changes, with another new one inserted. Furthermore, there are now rows with indentation, which indicates that that result has the same error stack trace as the first row above with no indentation.

The columns in the table are:

- **Status:** Status of the test result, whether it is passed, failed, or having an unknown status.
- **Test Case ID.**
- **Test Case Name.**
- **Stack Trace:** Stack trace from the execution environment for the cause of the failure.
- **Date Added.**
- **Bug ID:** ID of the bug causing the failure, if applicable.
- **Note:** Miscellaneous information regarding the test result.

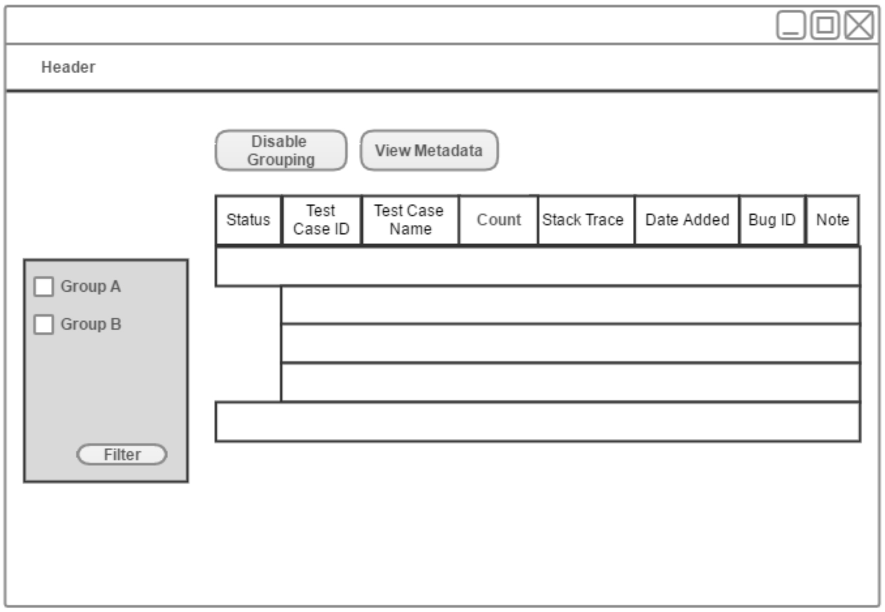

Figure 4-5 Test Result View Page - Grouped Mode

4.2.3 Test Result Metadata Page

In this page, users can see more detailed information regarding a test result, as well as adding, editing, or deleting some of those information. The design of this page is shown in Figure 3-6. Grayed text fields are those that contain data that cannot be changed.

The status on the upper right corner of the web page will either have 3 options: "Passed" in green color, "Failed" in red color, and "Unknown" in black color. If there are bug ID and/or note already inserted, their text field and text box will contain the stored data. Otherwise, they will be blank. Users can add or edit both of them, and once finished clicking "Save" button will store all the information into the database, and return the user the Test Result View Page. "Cancel" button will also return user to Test Result View Page, though no information will be saved.

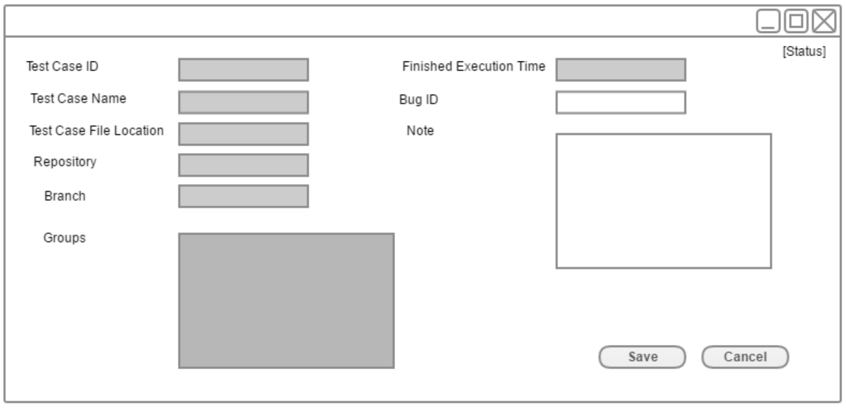

Figure 4-6 Test Case Metadata View Page

4.2.4 Running Cases View Page

This page is a simple one, only containing the common header, and a single table containing the list of test cases till being run in the execution environment. The design is described in Figure 3-7.

The columns in the table use the same data parameters as the ones that users insert in Text Execution Page. There are wo new columns, “Date Added” which shows the time this case is added, and “Status”, showing whether the test case is still in queue, or already running in the execution environment.

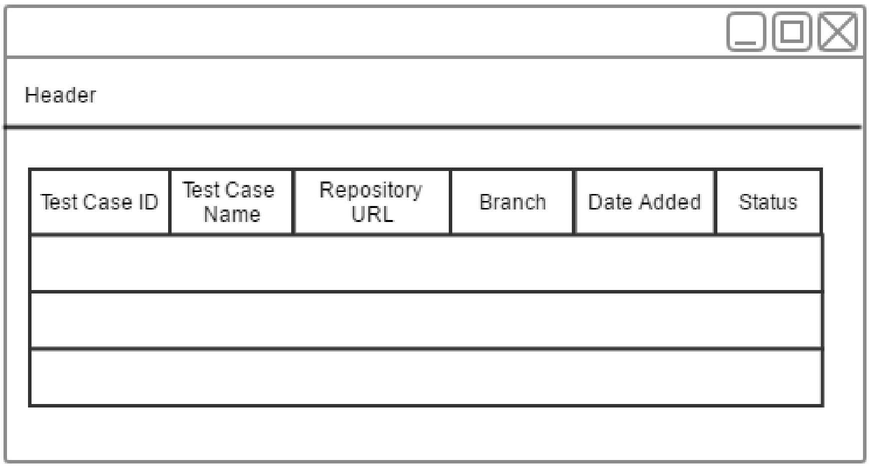

Figure 4-7 Running Cases View Page

4.2.5 Failures Management Page

This is also a straightforward page. This page groups all stack traces, and then list them with the total count of test results having that error with an option to enable auto re-execution. This page is intended

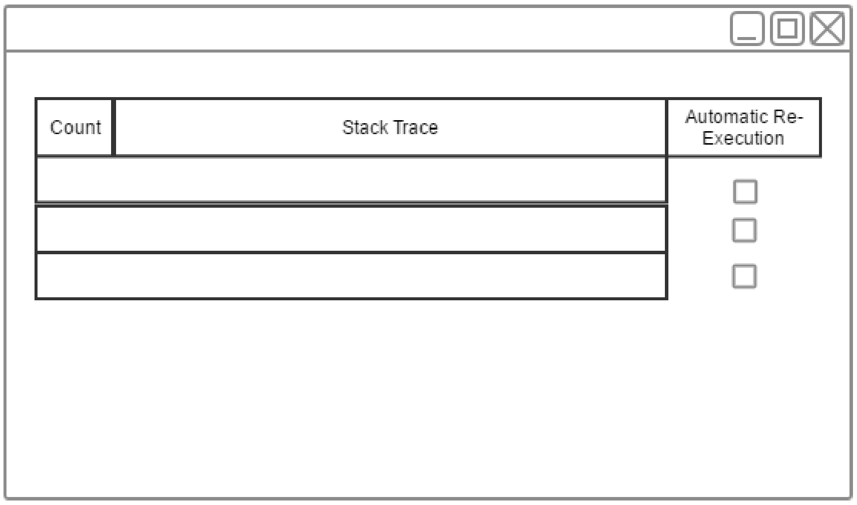

Figure 4-8 Failures Management Page

so that the system needs less human interaction to automatically resolve environmental errors (e.g. due to unstable network). The page is described in Figure 3-8.

4.3 Database Design

Considering the small amount functions that the current design has, it is more than sufficient to have simple entities that cover all of the functions. This also keeps with the design’s main objective of keeping it easy to develop and maintain.

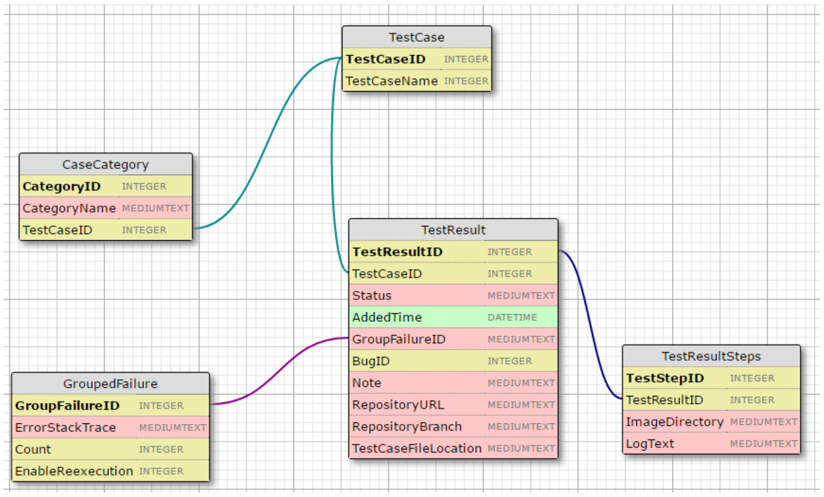

Figure 4-9 Database Schema

Whenever the system receives a test result from the execution environment, the system automatically processes it, and saves all information to the corresponding entities. This is just representation of the logical structure of the schema; it doesn’t show the underlying hardware specification, such as what RAID configuration the storage device uses.

Furthermore, the database doesn’t use any stored procedures nor triggers; all of such functionalities will be implemented into the data access management itself inside the system. This is done in order to limit the elements of the whole system that needs to be modified whenever a change needs to be made, making maintenance easier.

5 IMPLEMENTATION

From all of the main functions and the logic flows in the whole system, there are several important logic flows contained in the main functions that make this system unique from others. Using source code to describe those flows will take too many space, and thus they will be described using diagrams.

All of the diagram will use Swimlane diagram to show how the commands and decisions happen in the layers of the system, as described in section **3.1.2.**

5.1 TEST EXECUTION

This logic flow happens when an automation tester executes a test case. The diagram in Figure 4-1 describes how the system takes decision and call functions.

5.2 TEST RESULT RECEIVE

This logic flow happens when the system receives a test result from the test execution environment. The diagram in Figure 4-2 describes it.

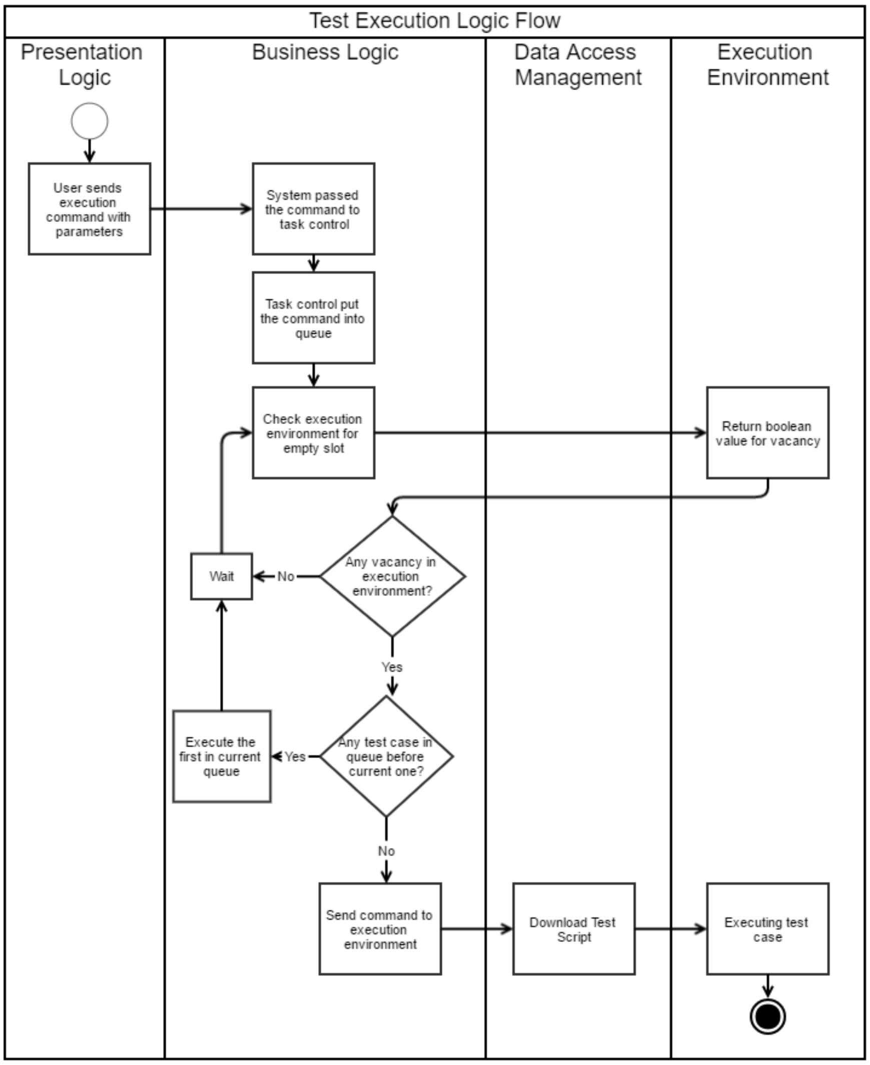
Test Execution Logic Flow
Presentation Logic
Business Logic
Data Access Management
Execution Environment
User sends execution command with parameters
System passed the command to task control
Task control put the command into queue
Check execution environment for empty slot
Return boolean value for vacancy
Any vacancy in execution environment?
No
Wait
Yes
Any test case in queue before current one?
Yes
Execute the first in current queue
No
Send command to execution environment
Download Test Script
Executing test case

Figure 5-1 Test Case Execution Logic Flow

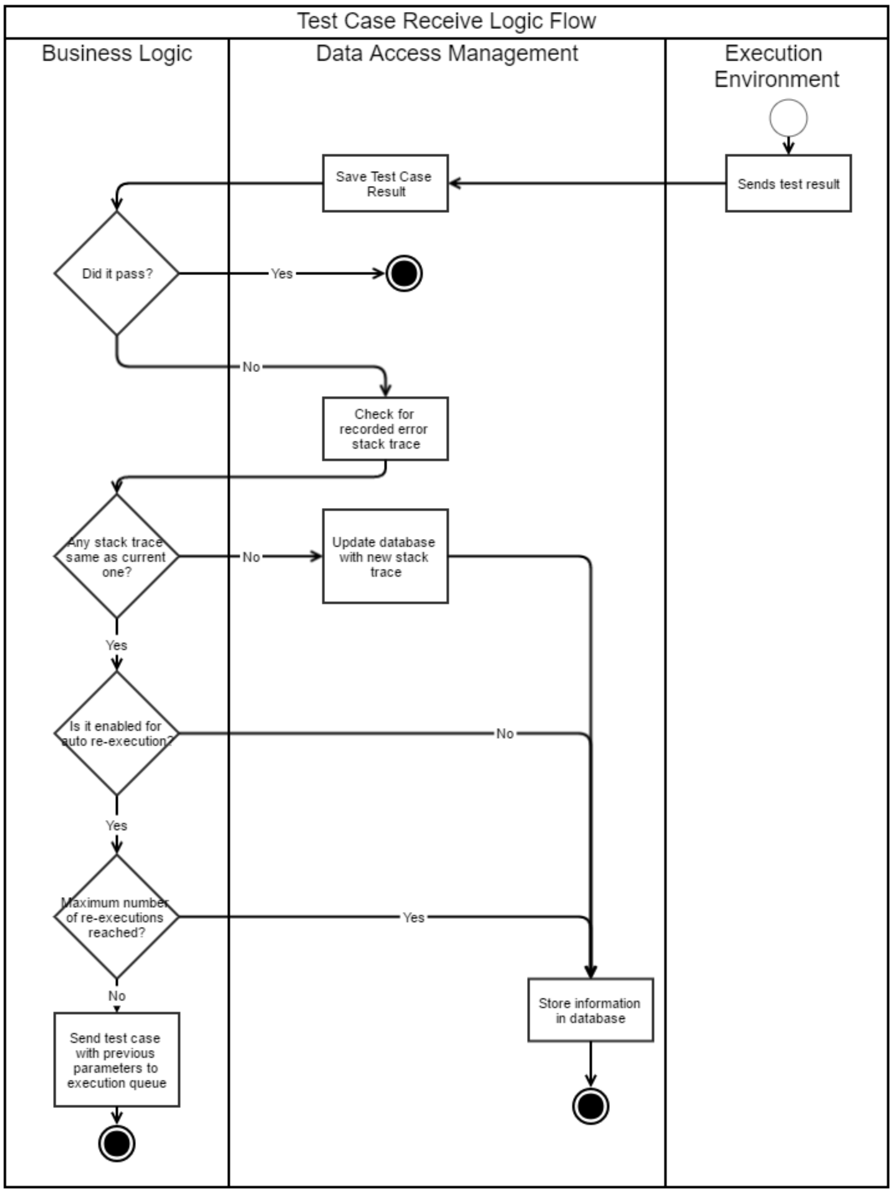
Test Case Receive Logic Flow
Business Logic
Data Access Management
Execution Environment
Sends test result
Save Test Case Result
Did it pass?
Yes
No
Check for recorded error stack trace
Any stack trace same as current one?
No
Update database with new stack trace
Yes
Is it enabled for auto re-execution?
No
Yes
Maximum number of re-executions reached?
Yes
No
Send test case with previous parameters to execution queue
Store information in database

Figure 5-2 Test Case Receive Logic Flow

6 TEST AND VALIDATION

In order to verify that the system designed fulfills the original design, tests must be done to validate the main objective of this software.

6.1 METHODOLOGY

As this system is intended for testing a software, the method used to test this system's outcome is by performing three same test cases manually, using Selenium for testing, and by using this system, and then comparing their results.

For each of the test case, it is repeated until achieving 3 passed run with each method, and then calculate average time taken for each method to pass each test case. Besides the actual time taken for the test to complete, the total time needed to develop the test scripts are also taken into account. The details of the test scripts, as well as the tested web page, can be referred from Appendix A.

6.2 RESULT

After running the test cases in 3 different methods, I achieved the result as pictured in Figure 5-1. There is no common trend between the test cases, as in test case 1 the manual method is 4 seconds longer than the rest. In test cases 2 and 3, there are very small differences in total time taken between the different methods.

In test case 1, the relatively huge difference is caused by the test case requiring many data input, and automated ones can do that much quicker than a person can do. For test case 2, automated ones take slightly longer time maybe due to the fact that there are steps inside the test script to wait for elements to be loaded, while actually they can already be operated. If the test cases are run more times using a very fast connection, probably there will be no differences.

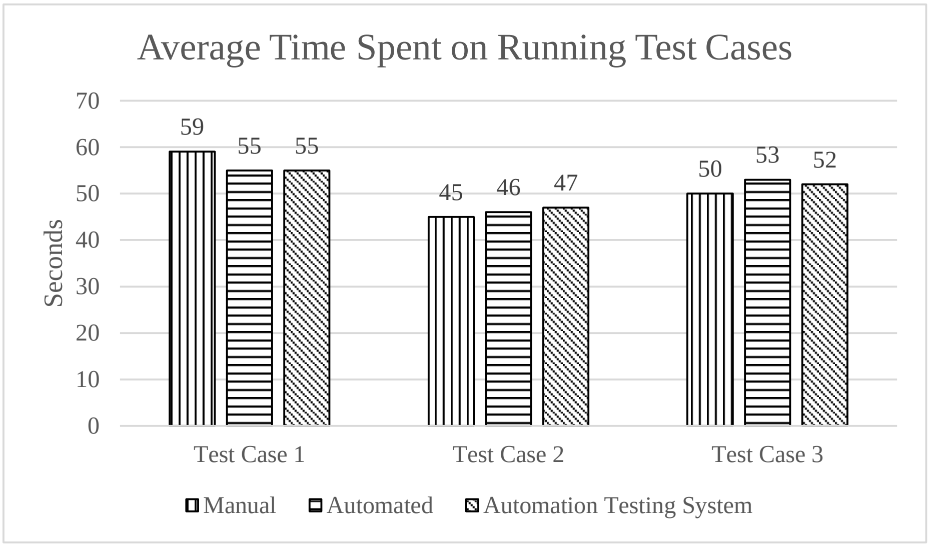

Figure 6-1 Averate time spent on running test cases

For the next measurement, the total time spent in developing the test script, does not apply to manual method since testers can immediately begin running the test case.

Across all test cases, using automation testing system takes much less time than Selenium in writing test script, as seen in Figure 5-2. This is due to the fact that SeLion, the core of this system, has many commonly used functions encapsulated, allowing easier and faster development. Besides that, SeLion has the ability to automatically generate a PageObject[13] based on YAML[14] file containing the element locators. In Selenium, that PageObject must be created manually.

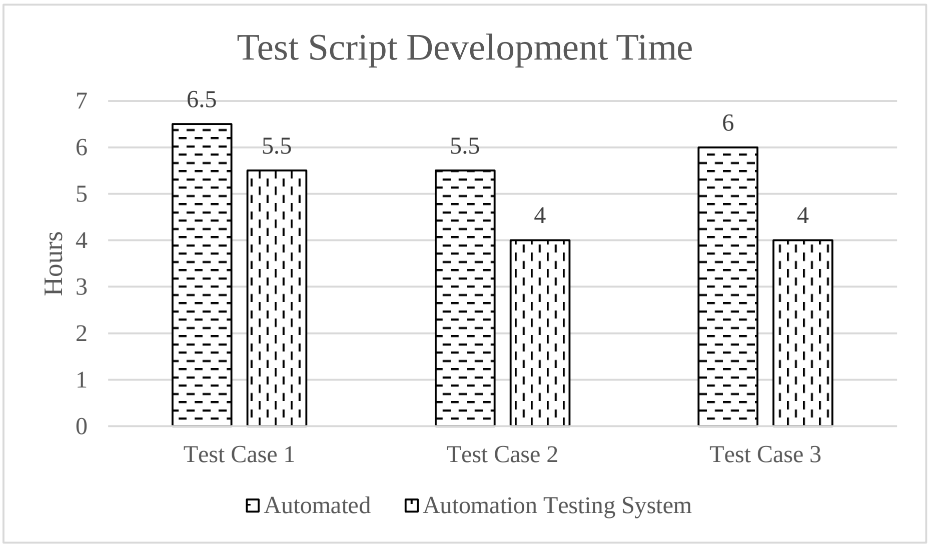

Figure 6-2 Test script development time

Furthermore, based on data taken from OECD, the average annual working hours in OECD members is just 1770 hours [17]. When compared to total amount of hours in a year, there is huge amount of free time. That much time can be used by the automation control system, as it supports queueing of tasks. Before any end of working day or any long term holiday, automation testers can queue a lot of tasks, so that when they come back they can get test results ready to be analyzed. This is so much efficient than manual testing, which can only be done during working hour, and also normal automation testing, which doesn’t support queueing.

---

[13] PageObject = A Java class that expresses a web page as a Java object.
[14] YAML = Yet Another Markup Language.

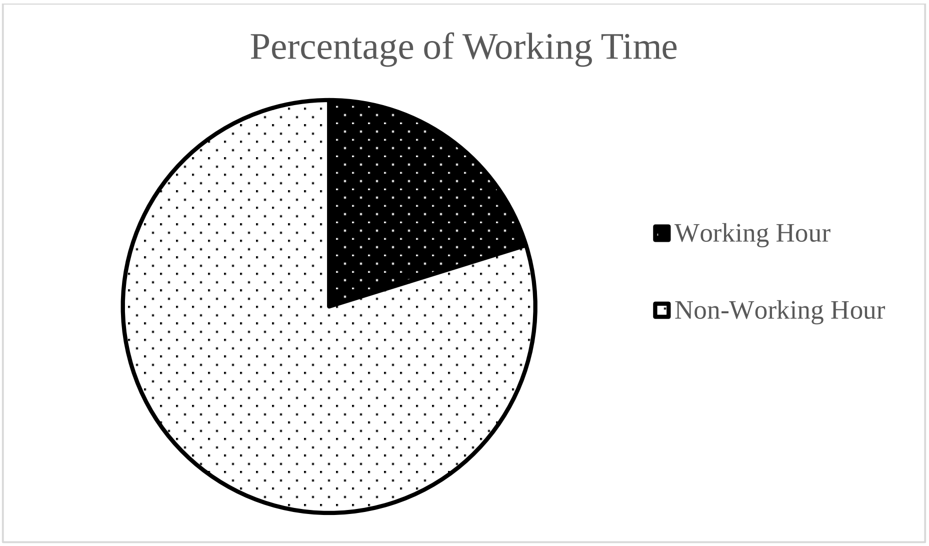

Figure 6-3 Percentage of Working Time in Working Days

6.2.1 Impact on Maintenance

In many software, features have flow structured like a tree; they begin from a same starting point, and then move to different features and sub features. Many features have similar parts, or slightly modified version of parts of other test cases.

Like in these test cases, when comparing between test case 2 & 3, more than half of the steps are similar, just a matter of different parameters or element locators. In automation testing, if there is a finished & reviewed test case containing steps needed by a new one, automation tester can just copy and modify that part to be used in the new test case.

This significantly reduces overall time needed to develop a test script, as reviewed test case is almost guaranteed to be working and bug free and thus less time spent in debugging a test script. This effect snowballs as number of test cases grow because many test cases contain similar steps, though everything must be reviewed whenever the tested product receive major changes.

Time taken to develop the next test script will decrease, and because running the test scripts can be done outside working hour, automation testers do not need to spend time waiting for their test cases to finish. These ultimately saves a lot of time, reducing the time required to perform testing on a software. Additionally, as it takes less time to finish a test script, automation testers can cover more scope of the product software compared to manual tester.

6.2.2 Impact on Expansion

When software adds supports more language and/or country, or add new functionalities, they are considered to be software expansion. Testing the software on different language will be difficult, since the tester team must have someone who speaks that language in order to be able to understand the GUI.

A well written automation test script can be used immediately to test the software even on a different language than the original one; a good test script does not use any labels as a locator, since they change often. A good test scripts relies entirely on the structure of the page, such as using XPath or CSS locator in web page, which rarely change between different languages of the same page.

Thus, when a new language is supported, almost all existing test cases can be immediately used without problem, with few exceptions such as those involving with country specific data, such as address format. Besides that, when a software is made publicly available in another country, occasionally parts of it needs to be changed to abide to the local law, which results in modifications in existing test cases to accommodate all variations within the same step.

Points about how automation testing saves time on new features have been made in previous section (section **5.2.1**), and with the advantages it also brings to language expansion, automation testing saves a considerable amount of time compared to manual testing when performing tests on a software expansion. Assuming that the new language doesn’t have any special steps, supporting a new language on an existing test case can take less than an hour, as it is only a matter of new set of parameters.

When test cases for new features and languages get finished quickly, the development team can begin using them to do regression testing as they develop the software, which will increase the final quality of the software while taking less time to complete.

7 CONCLUSION

As today’s software gets more and more complex, while being developed in new development style which demands constant stream of small changes, the method used for testing the software must change too. This is very important for business oriented software and software that provides service to consumers.

Typical automation testing framework by itself doesn’t have the necessary functionalities to be fulfill all the demands of a software project. Thus, a new system is made to complement SeLion to add further functionalities. SeLion by itself already has many functions for performing automation testing.

The automation testing control system will be entirely developed using open-sourced technologies in order to keep development easy and low cost, as well as allowing a lot of customization in the future.

All things considered, when testing a single test case, there is almost no difference in total time taken between manual test case, automated one, and the one using this system. However, when the system is used for long term, the amount of time and effort saved is tremendous, especially in software maintenance and expansion.

Though the system doesn’t use the latest, highest performing technologies, using Java and technologies based on it is more than enough for the system as the environment assumed to host the system is not restricted in terms of computational power. Keeping things simple make the system easy to develop and maintain.

Thus, this system is recommended for software development team large enough that developing and maintaining this system won’t take a huge chunk of its total capability, and for teams that are working on or planning to work on a large software project that lasts for years.

8 REFERENCES

[1] A. Kolawa and D. Huizinga, "Automated Defect Prevention: Best Practices in Software Management.," *Wiley-IEEE Computer Society Press,* p. 74, 2007.

[2] VersionOne, Inc., "7th Annual State of Agile Development Survey," 2013.

[3] T. Tamai. [Online]. Available: http://tamai-lab.ws.hosei.ac.jp/pub/icsm92.pdf. [Accessed 16 March 2016].

[4] G. Yardley, "Slide Share," pinch media, 18 February 2009. [Online]. Available: http://www.slideshare.net/pinchmedia/iphone-appstore-secrets-pinch-media. [Accessed 16 March 2016].

[5] T. M. Pigoski, Practical software maintenance: Best practices for managing your software investment., New York: Wiley Computer Pub., 1997.

[6] Economist Intelligence Unit, "Closing the gap: The link between project management excellence and long-term success," 2009.

[7] J. Gosling, B. Joy, G. Steele, G. Bracha and A. Buckley, "The Java® Language Specification," 2015.

[8] Oracle, "The Java Language Environment," 1997. [Online]. Available: http://www.oracle.com/technetwork/java/intro-141325.html.

[9] Oracle, "Differences between Java EE and Java SE," 2012. [Online]. Available: http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.

[10] Oracle, "What is MySQL?," 2016. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html. [Accessed 27 April 2016].

[11] The Apache Software Foundation, "Apache Tomcat®," 2016. [Online]. Available: http://tomcat.apache.org/. [Accessed 27 April 2016].

[12] R. S. Pressman, in *Software Engineering A Practioner's Approach*, 7th ed., McGrawHill, 2013, pp. 308-309.

[13] IEEE Standards Association, "982.1-1988 - IEEE Standard Dictionary of Measures to Produce Reliable Software," 1988.

[14] Akamai, "Akamai Reveals 2 Seconds As The New Threshold Of Acceptability For ECommerce Web Page Response Times," 2009. [Online]. Available: https://www.akamai.com/us/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp. [Accessed 30 April 2016].

[15] ISO, "ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability," Geneva, 1998.

[16] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?," *FAST'07: 5th USENIX Conference on File and Storage Technologies,* pp. 14-15, 14-16 February 2007.

[17] OECD, "Average annual hours actually worked per worker," [Online]. Available: https://stats.oecd.org/Index.aspx?DataSetCode=ANHRS. [Accessed 4 May 2016].

9 TABLE OF FIGURES

Appendix A

Test Cases for Automation Testing Control System

Case 1:

Website:	https://ludeon.com
Description:	Register, then create and preview a thread post
Goal:	Preview a thread post
Steps:	1 Open the website 2 Access Forums 3 Click on Register 4 Agree to Terms & Conditions 5 Fill in all necessary data, and then register 6 Click on “General Discussion” 7 Click on the first forum topic 8 Click reply 9 Type any 100 keyboard characters 10 Click preview

Case 2:

Website:	www.jd.com
Description:	Book a domestic flight ticket
Goal:	Login prompt page
Steps:	1 Open the website 2 Click on flight ticket link 3 Choose domestic flight 4 Enter destination city 5 Pick departure & arrival date 6 Choose returning flight 7 Click search 8 Click on the first booking button

Case 3:

Website:	www.jd.com
Description:	Book an international flight ticket for 3 adults

Goal:	Login prompt page
Steps:	1 Open the website 2 Click on flight ticket link 3 Choose international flight 4 Enter destination city 5 Pick departure & arrival date 6 Choose returning flight 7 Open advanced tab, choose 3 adults 8 Click search 9 Click on the first booking button