

**REPORT ON**

**PROFESSIONAL INTERNSHIP**

**WITH**

**AUTODESK SINGAPORE R&D**

PREPARED BY:

DOAN DUY DUC

U1440085B

Computer Engineering

**Content**

Abstract 3

Acknowledgment 4

List of symbols and abbreviations 5

1. Introduction 6

1.1. About the company 6

1.2. About the project 6

2. Project purpose and characteristics 7

2.1. Project purpose 7

2.2. Project characteristics 7

3. Stages of Development 9

3.1. Migrate code 9

3.1.1. The need of code migration 9

3.1.2. Overview of project flow after migration 9

3.2. Add more functionalities to the framework 10

3.2.1. Add project management tool 10

3.2.2. Add Selenium Grid to the framework 11

3.2.3. Add Plug mechanism 11

3.2.4. Modify input method and parameters 12

3.3. Unit testing and Documentation 12

3.3.1. Unit Testing 12

3.3.2 Documentation 13

Conclusion 14

Appendix A: Selenium WebDriver 15

Appendix B: Selenium Grid 16

Appendix C: Maven 17

**Abstract**

This report provides a summary of my individual training and industrial experiences during Professional Internship at Autodesk Singapore R&D. During this period, I have been involved in a project using Java programming language to build a Web Automation Framework to test localized UI bugs. Until now, the migrated framework in Java is stable and can be used to serve the original purposes of testing Autodesk documents and open for further extension to test Autodesk web applications. Throughout the time of this internship, I have opportunities to work with professional software developers and enhance my coding as well as interpersonal skills.

**Acknowledgement**

I would like to express my gratitude to many people who have been giving advices, support and helping me during the internship period as well as proofreading this report.

I would like to thank Mr. Pradip Patil, senior software developer at Localization Solutions team in Singapore, my Autodesk supervisor, who has been giving many advices during the process of building this project and also proofread this report.

Thanks to Mr. Marco Ganci, senior software developer at Localization Solutions team in Switzerland, who worked closely with me in this project and I have learnt a lot programming skills as well as software project design from him.

Last but not least, I want to thank all the people from Localization team, other interns and anyone that I have a chance to meet or work with during my time at Autodesk for such great experiences.

**List of symbols and abbreviations**

ASRD – Autodesk Singapore Research and Development

Chame10n – Project Title

UI – User Interface

**1**

**Introduction**

* 1. **About the company**

Autodesk, founded on January 30 1982, is an American multinational software company that builds support tools for architecture, engineering, construction, manufacturing, media, and entertainment industries [1]. Most popular products of Autodesk include: AutoCAD, Revit, Inventor, Remake, etc.

In Singapore, Autodesk R&D was established on ……… with hundreds of employees.



*Figure 1. Autodesk logo*

* 1. **About the project**

My project title is *Chamel10n*, it is a Web Automation Framework with the purpose of testing Web UI to find localized bugs such as text truncation, missing spaces, broken links, extra symbols, etc. More functionalities also need to be added so that performance is improved.

In this report, I will summarize the process of developing this project and along with all the experiences that I have gained. Main contents include:

* Project purpose and characteristics
* Stages of project development
* Current process and achieved results
* Things to do with the project in the future
* Key takeaways from the project

**2**

**Project Purpose and Characteristics**

* 1. **Project Purpose**

Chamel10n is an extension of a current automation framework written in C# that tests the localized deflections in Autodesk web documents (e.g. online tutorials of Autodesk products). To extend the working scope and take advantage of current functionalities, the project would be migrated to Java to increase compatibilities with web applications. Moreover, once the migration is completed, significant generalization at high level must be made in order to include general Autodesk web applications testing in the framework, instead of just Autodesk documents.

* 1. **Project Characteristics**

After analysis, the project should have the following characteristics:

* *Pluggable*:

The project must be able to easily manage different type of applications using adapter-based mechanism. Adapter is in charge of processing the input information, managing the language switch and executing pre-operations on the page based on web/cloud application type. For example, we could exclude the left side menu from the test case when it is the same for all pages.

* *Reusable*:

All the test cases in this project must be application independent, written once, run against all products. This follows the basic rule of software design that a project should be open for extension.

* Language Independent:

The framework including adapter, test cases, etc. must be well generalized in order to run the test suite against all supported languages. Users of the framework can specify which language they want to run and the tests should adjust accordingly.

* Flexible:

As different issues may arise in the localized pages, multiple test case types must be supported. Test cases can be categorized into two main types

* Scan Test case: Scan single page to find format errors such as broken links, extra symbols, missing spaces, wrong links, etc.
* Compare Test case: Compare source page in English with localized pages to make sure that the translation process does not break the presentation the pages. Some examples are: missing content, missing images, missing links, untranslated contents, etc.

**3**

**Stages of Developments**

* 1. **Migrate code**
     1. **The need of code migration**

As introduced in section **1.2.**, this project is an extension of an existing framework in C# with functions of testing Autodesk documentations. In the initial stage, the code must be translated from C# to Java before adding more features. The final purpose of this stage is to create a framework in Java that behaves and produce the same results as the original projects in C#.

Both C# and Java are strong object oriented programming languages and, hence, have many features, syntaxes as well as problem approaches in common. However, there are also many unique packages, methods, functions that exists only in C#. In this case, researches should be made to find compatible packages in Java that have the same expected behaviors. If such packages could not be found, manual factorization and new methods must be created in order to provide the same functions.

As the code base is quite large, this initial stage would take quite significant amount of time.

**3.1.2. Overview of project flow after migration**

The purpose of the framework is to find localized bugs in Web UI. In order to achieve this, the work is divided into four stages:

* User Input: the user inputs are provided in multiple .ini files. Important inputs include:
* Specific information to identify the urls to the pages that need to be tested
* Languages: A set of languages to be used for generating the test page as well as general use in the testing process
* Web-specific information: each supported Autodesk web application/document has its own input section to specify initial configuration before testing
* Directory information: the path to find input files, project roots, etc. are specified in this section.
* Adapter: each web/document type has its own adapter to configure the test engine before running the test cases. The adapter is selected and initiated based on the corresponding sections from user input.
* Test case: after initial configuration, the logic of the test cases is executed. To increase the efficiency and reduce run time, the test cases are divided into threads and run at the same time. However, this is still limited to the capacity of one running machine. In this section, Selenium WebDriver (view *Appendix A* for more information) is used to automate the testing process. Selenium is in charge of navigating to the pages, filling in input boxes when necessary, grabbing text data from the page for testing, etc.
* Logging: All the results of testing process are recorded in testing logs. The log contains information about project configuration, tested pages, outcome of the test cases, exceptions if happened, running time, etc.

**3.2. Add more functionalities to the framework**

**3.2.1. Add project management tool**

As the new project is written in Java, Maven automation tool is added to manage the external packages used during the process of writing new Java code. Moreover, Maven also helps automate the testing process as user can schedule which activities to be executed. For example, with one Maven command, we can execute the framework and then run unit tests as well as generate Javadoc. This helps save framework development and testing time.

More on Maven in *Appendix C*

**3.2.2. Add Selenium Grid to the framework**

As in the process of executing test cases, the test cases are distributed into threads and run at the same time. Moreover, the number of threads has to be hardcoded in order to create the thread pool. This is also limited to the capacity of one running machine (localhost machine).

This issue can be solved with the help of Selenium Grid (more on Selenium Grid on *Appendix B*). When the grid is properly set up and configured, the test cases can be run against different environment at the same time. In addition, as multiple machines in the grid are used, more threads can be invoked simultaneously leading to better runtime performance. For example, for Chrome driver, the default number of drivers that can be used at the same time is 5, which means for 10 machines, user can run 50 test cases at the same time. Based on the result obtained in the log, performance time are approximated recorded in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 test case | 10 test cases | 100 test cases |
| 1 machines | 3 minutes | 30 minutes | 150 minutes |
| 10 machine | 3 minutes | 5 minutes | 7 minutes |

It is can be deduced that with large enough grid, the worst runtime of all test cases can be roughly approximated as the running time the one test case that takes the longest time. In practice, the real performance is significantly improved as expected.

**3.2.3. Add Plug mechanism**

The idea of adding plug mechanism is proposed in order to further optimize the testing process. Plug is in charge of doing some business logic after the adapter is configured and before the test cases are executed. The plug is selected to open dialog boxes, prepare text, containers and other resources that are used in testing process. This feature is still under development.

**3.2.4. Modify input method and parameters**

The framework after migration uses INI files and input files are dispersed in different directories. In java project, all the inputs are kept in one unique JSON file. This configuration files contains all sections that specifying the original necessary information such as User Input, Adapter configuration, etc.

In addition, as the Selenium Grid and Plug mechanism are added to the framework, new parameters also need to be added. For Selenium Grid, user can specify whether they want to use the grid or just use the local machine, number of machines in the grid to use, environment to run the tests against (browser, operating system), etc. For plug mechanism, user can also specify if they want to run the plug logic before executing the test cases or configure some parameters to run the plug logic.

In this project, JSON files are read using org.json package to read and update JSON file.

**3.3. Unit testing and Documentation**

**3.3.1. Unit testing**

Along with developing new features of the framework, each class or method needs to be tested in order to make sure they would still work under any changes in the framework. This helps debugging much more convenient and faster and saves a significant amount of manual testing as the project grows and new components are added.

In this project, unit testing engine used is Junit. It is a simple framework to write repeatable tests in Java.

**3.3.2. Documentation**

For every project, documentation is an essential part. All classes, methods purposes, parameters and expected outcome are automatically generated with Javadoc. However, in order to create the doc as expected, one needs to add comments to every class and method in correct format. Documentation could be generated once all the features are complete or created temporarily for better view of current structure of the project.

**Conclusion**

This project provides me with a chance to be involved in every stage of software development process from management, design to testing and documentation. Along the way, I have learnt many new technologies that are applicable not only in the scope of this project but also in my future projects.

Moreover, during the time working in this project at Autodesk, I also sharpen my teamwork skills when collaborating with senior developers as well as fellow interns.

Last but not least, working at a professional industrial environment benefits me a lot in a way that I could ask questions and receive responses and advices from senior developers. All the code is thoroughly reviewed and refactored when necessary. Therefore, I have a chance to apply knowledge at school about object-oriented programming and software design pattern to achieve flexible and extensible solutions.

**Appendix A: Selenium WebDriver**

Selenium WebDriver accepts commands and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser, and retrieves results.

Unlike in Selenium 1, where the Selenium server was necessary to run tests, Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. However, Selenium Grid can be used with WebDriver to execute tests on remote systems (see below). Where possible, WebDriver uses native operating system level functionality rather than browser-based JavaScript commands to drive the browser. This bypasses problems with subtle differences between native and JavaScript commands, including security restrictions.

In practice, this means that the Selenium 2.0 API has significantly fewer calls than does the Selenium 1.0 API. Where Selenium 1.0 attempted to provide a rich interface for many different browser operations, Selenium 2.0 aims to provide a basic set of building blocks from which developers can create their own [Domain Specific Language](https://en.wikipedia.org/wiki/Domain_Specific_Language).

As of early 2012, Simon Stewart (inventor of WebDriver), who was then with Google and now with Facebook, and David Burns of Mozilla were negotiating with the [W3C](https://en.wikipedia.org/wiki/W3C) to make WebDriver an internet standard. In July 2012, the working draft was released. Selenium-WebDriver (Selenium 2.0) is fully implemented and supported in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), and [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)).

In this project, Selenium is written in Java. Configuration is written from JSON file and the corresponding web driver is invoked accordingly.

**Appendix B: Selenium Grid**

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

The ability to run tests on remote browser instances is useful to spread the load of testing across several machines, and to run tests in browsers running on different platforms or operating systems. The latter is particularly useful in cases where not all browsers to be used for testing can run on the same platform.

In this project, the grids are created with virtual machines which are pre-configured to act as hub or node. User of the framework just needs to provide the information of the hub and the grid will adjust itself and find out how to distribute the test case thread to different node machine.

**Appendix C: Maven**

Maven is a tool for building and managing any Java-based project. Maven’s primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

* Making the build process easy

While using Maven doesn’t eliminate the need to know about the underlying mechanisms, Maven does provide a lot of shielding from the details.

* Providing a uniform build system

Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system. Once you familiarize yourself with how one Maven project builds you automatically know how all Maven projects build saving you immense amounts of time when trying to navigate many projects.

* Providing quality project information

Maven provides plenty of useful project information that is in part taken from your POM and in part generated from your project’s sources. As Maven improves the information set provided will improve, all of which will be transparent to users of Maven.

Other products can also provide Maven plugins to allow their set of project information alongside some of the standard information given by Maven, all still based on the POM.

* Providing guidelines for best practices development

Maven aims to gather current principles for best practices development, and make it easy to guide a project in that direction.

For example, specification, execution, and reporting of unit tests are part of the normal build cycle using Maven. Current unit testing best practices were used as guidelines:

Maven also aims to assist in project workflow such as release management and issue tracking.

Maven also suggests some guidelines on how to layout your project’s directory structure so that once you learn the layout you can easily navigate any other project that uses Maven and the same defaults.

* Allowing transparent migration to new features

Maven provides an easy way for Maven clients to update their installations so that they can take advantage of any changes that been made to Maven itself.

Installation of new or updated plugins from third parties or Maven itself has been made trivial for this reason.

**References**

[1] Wikipedia, Autodesk <https://en.wikipedia.org/wiki/Autodesk>

[2] Wikipedia, Selenium <https://en.wikipedia.org/wiki/Selenium_(software)>

[3] Maven introduction, [https://maven.apache.org/what-is-maven.html](https://maven.apache.org/what-is-maven.html%20)