City University of Hong Kong
Department of Computer Science

BSCCS Final Year Project Report 2014-2015

14CS187

A system to facilitate test automation development for web applications

(Volume 1 of 1)

Student Name : ZHOU Anqi
Student No. : 52165074
Programme Code : BscCS

For Official Use

Supervisor : Dr YU, Yuen Tak
1st Reader : Dr CHAN, Wing Kwong
2nd Reader : Prof JIA, Xiao Hua

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

A system to facilitate test automation development for web applications

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: ZHOU Anqi Signature: _____

Student ID: 52165074 Date: 09 June, 2015

ABSTRACT

Graphic user interfaces (GUIs) are constantly adopted by many software applications to provide a visual connection between the user and the underlying components of the application. Over the decades, considerable researches have been undertaken in the following three processes to reduce the testing effort: the test case generating process, the test case executing process, and the maintenance process. Automating these testing processes can save repetitive human efforts and prevent human errors especially when applications are widely developed by iterations nowadays. However, developing automation script is always associated with increased effort, which can be expensive in programmer time and a common problem is how this increased effort can be reduced. This project will base on the capture and replay tool Selenium IDE and we propose three major techniques that facilitate the tester to generate a maintainable large-scaled automated test suite without writing actual code and with reduced effort. The first two techniques namely crawling technique and script formatting technique are to convert the messy scripts generated by Selenium IDE to a well-structured and maintainable test suite adopting page object pattern by extending the functionalities of Selenium IDE. The third technique is motivated by feedback-directed random testing algorithm and adaptive random testing to generate test inputs based on the existing test suite. There are also other supplementing functionalities in this project like generating test data, importing user data to ease the automation script development.

Table of Contents

Chapter 1 INTRODUCTION

1.1 Motivation & Background Information

Graphic user interfaces (GUIs) are constantly adopted by many software applications to provide a visual connection between the user and the underlying components of the application. Thus, the importance of GUI testing to provide user correct behavior of applications is widely acknowledged (Cai et al. 2005). Generally, GUI testing is the process where the test case is generated and conducted in a sequence of events and test result is compared with the expected result (Memon et al. 2002).

Over the decades, considerable researches have been undertaken in the following three processes to reduce the testing effort: the test case generating process, the test case executing process, and the maintenance process. The most noticeable challenge is the test case generation process for web applications. Compared with traditional desktop applications, web applications pose several unique qualities make them more challenging (Li et al. 2014). Web applications typically consist of back end and front end, which can be implemented using multiple programming languages such as JavaScript, C#, HTML, CSS, PHP and so on. Secondly, web application is more vulnerable to various attacks due to its open operating environment. Moreover, web applications support multiple users and the multi-threaded nature of web application also makes it more difficult to detect and reproduce the errors. Last but not the least, web application technologies and frameworks are fast evolving requiring testing techniques stay current. Sedar et al. (2014) have conducted a systematic mapping study and a systematic literature review on the papers proposing various techniques for testing web applications. Li et al. (2014) have further presented a survey on web application testing advances in the past two decades. In particular, they have conducted a comprehensive review on recent popular testing techniques, including graph and model based testing techniques, mutation testing techniques, search based testing techniques, scanning and crawling testing techniques, random testing with the use of assertions as the primary oracle, black box and white box fuzz testing, user-session-based techniques. In section 2.2, we will discuss some of the above techniques

and related works motivating to this project and we can adopt part of their technical merits in this project.

At the same time, test engineers also employ some automation tools (e.g., Selenium, QTP and waiver) to automate the testing execution process with scripting while some utilize capture and replay tools (Stanislava and Bernardino, 2009) which work by noting the mouse motion and keystrokes where test scripts can be recorded automatically. Moreover, there are also advances in automating test suite repairing process. For example, Huang et al. (2010) employed genetic algorithm to repair GUI test suite. This project will mainly focus on reducing human effort in these three processes: test case executing process, and the maintenance process and test case generation process in web application GUI testing.

Admittedly, utilizing tools to automate the testing process with scripting does save repetitive human efforts and prevent human errors especially when applications are widely developed by iterations nowadays. However, developing automation script is always associated with increased effort, which can be expensive in programmer time and a common problem is how this increased effort can be reduced. Capture and replay tool can save the effort by generating scripts automatically. However, capture and replay is designed to be a quick solution to automation, not a solution to a full regression test suite where the generated script is messy and unclean, which, could result in costly maintenance of the test suite. Further, this kind of tools usually lack more advanced functions such as multiple browser support. To tackle this problem, Uppal et al. (2012) have developed an extension for a capture and replay tool - Selenium Integrated Develop Environment (IDE). Selenium IDE is a Firefox plugin, which records user actions as a list of commands while Selenium web driver is an application interface (API) aiming at creating a robust and scalable test automation using programming language such as Java, Ruby. And they combined Selenium IDE and Selenium web driver to provide multiple browsers support for the generated script. Nonetheless, the output scripts are still hardcoded with bad code practice and given the automation tool, constructing test case remains a resource demanding activity if done manually.

This project will base on the capture and replay tool Selenium IDE and we propose three techniques that facilitate the tester to generate a maintainable large scaled

automated test suite without writing actual code. The basic idea of Selenium IDE is that it records every user-action as a command consisting of action, target and value and treats a test case as a sequence of commands. The input of this project is the web application and tester to capture a basic set of test suite using record and replay tool.

The first two techniques are to convert the messy scripts generated by Selenium IDE to a well-structured and maintainable test suite by extending the functionalities of Selenium IDE. Firstly, we will adapt crawling techniques, following the approach implemented in Manning et al. (2008) and Memon et al. (2003) to separate variables like the properties and attributes of user interface elements and user input data from scripting logic at capture time and store the elements in resource files. More recently, Mesbah et al. (2012) proposed a more advanced crawling technique for Ajax based web application, which are motivating to this project. The second technique is to group repeated event sequences together and save them as one function making the function reusable across one test suite and we will reconstruct the low level script output by selenium IDE to reuse the components by applying design pattern to test automation suite. Currently, we are applying Page-object pattern, which has been popular in the industry (Maurizio et al. 2013). Page-object pattern aims to provide abstraction between test cases and the application under test utilize a single view or page as an object which have properties and function. For example, the login page object has a login function can be reused in many scripts that contain the login procedure. If the login logic is to change, there is only one single point update.

The third technique is motivated by feedback-directed random testing algorithm and adaptive random testing to generate test inputs based on the existing test suite. The event driven execution nature of web application could result in tremendous test inputs using undirected random testing due to the variation of sequences of events and type of data. We introduce a framework following the merits of feedback-directed random testing and the adaptive random testing based on the existing test suite to generate the test input according to the existing test cases. Feedback-directed random testing can reduce the size of the candidate pool and eliminate the duplicated test cases while adaptive random testing can select the test inputs according to the distance of the input from the existing test suite, which increase the test coverage. We introduce a new framework, which takes the following as input: an execution unit

which is the browser, existing test suite. In section 3.2.3 and 4.2.5, we will discuss the algorithm and detail implementation.

Moreover, a test oracle is required to convert the test input to test case. We propose to adopt the similar approach in the (Mesbah & Van Deursen, 2009) using invariants on the DOM trees including generic DOM invariants and application-specific invariants as test oracles. These invariants derived from existing testing suite and invariants when executing the new generated test input could also be used as regression oracle.

1.2 Scope definition

We define our project scope mainly from three respects namely test levels, test activity and test locations. Test activities includes test case design, test automation, test execution and test evaluation (test oracle). Test level includes unit testing, system testing, integration testing. Test location includes front-side and server-side (Garousi et al. 2013, Dogan et al. 2014). In this project, we will focus on test case generation, test script generation and test case maintenance process. And we will review various testing techniques and other technique will be used to facilitate the project. Currently, this project is mainly for client side GUI testing for web applications. As this project is intended to be general and not depending on specific language. We will mainly focus on black box system testing without the need of the knowledge of the applications' source code.

Chapter 2 LITERATURE REVIEW

In this section, we would analyze different techniques in use and related work for proposed solution, evaluate existing applications, address the weakness, extract positive characteristics and summarize what approach should be chosen in the various technical options.

2.1 Web GUI testing

Web GUI consists of a set of web elements (e.gs. links, buttons, forms) that a user can perform a sequence of actions (e.gs. click, type) to interact with the applications. GUI provides an efficient way to use. However, implementing GUI always associates with a large percentage of code (Memon, 2002). Therefore, correctness of GUI is vital where complete and efficient GUI testing is needed. Memon(2002) pointed out that conventional testing involves the following steps: determine what to test by coverage criteria (a guideline for determining the completeness), generate test cases from specifications and requirements, determine expected result (test oracle) and execute test cases and verify the test result, determine whether test coverage is satisfied. And since the software is evolving, there is always a need to ensure the code change would not derive new errors and regression test is always needed to perform (Bauersfeld, 2013). Those result in testing process being labor-intensive and accounting for more than half of the cost of software development. As a result, many studies have been done to employ various techniques and tools to reduce the effort. We will discuss several techniques including graph and model based testing techniques, crawling testing techniques, random testing and concolic testing as these techniques are easier to automate the test case generation process and more relevant to the project. And we will present some related works, which are motivating to this project.

2.2 Extract GUI elements using crawling technique

In order to extract the web UI elements to separate data from logic, we are going to adopt the crawling technique to get the properties and value of UI elements and store them in a UI map where all elements locator are stored to ease the modification. Recently, research on mobile crawling has some advances (Amalfitano et al. 2011). However, we will mainly focus on desktop web application crawling.

Web crawling is the process of gathering as many web pages from the Web efficiently to index them, store the link structure to support a search engine (Manning et al. 2008). Figure 6 demonstrates the methodology behind crawling. The crawler starts with root url and get the context of the web page at that url using http protocol. Then the links and the text inside are extracted after the web pages parsed to do recursive crawling. Moreover, the links extracted will be tested through a series of procedures to determine whether should be added to URL frontier which is composed of URLs whose pages to be fetched. On the other hand, web crawling technique in search engine is only able to get text and links, not including buttons, images, input and so on. Furthermore, the traditional web crawling is not applicable to some dynamic web applications especially those with Ajax where usually user interacts with web application without URL change.

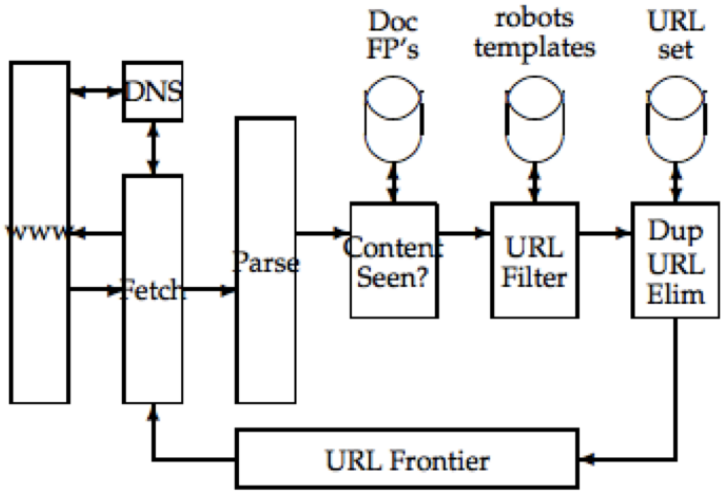


Figure 1 Basic crawler architecture (Manning et al. 2008)

As mentioned above, since the main objective of adopting crawling technique in this project is to get the UI elements of the web pages that the tester is interacting with and store them in Object Repository. Since the tester is involved in the tester design and capturing phase, the difficulties of crawling are much reduced. It is proposed to crawl the web pages the tester interacts with at the capturing time where the browser is

opened.

2.3 GUI test automation tools in web applications

Test automation eases the testing process which refers to the use of special software to automate software testing in contrast with manual testing in which all testing tasks are done manually. Test automation can improve the development process of a software product in many cases. There are some tools and framework currently used widely to automate GUI testing like commercial tool QTP, open source framework watir and Selenium, After comparison, Selenium is selected to serve as the base tool to realize the proposed solution

2.3.1 Quick Test Professional (QTP)

QTP is a dominant commercial test automation tool aiming at functional automation of Windows and Web based applications based on Visual Basic (VB) Scripting language. It can identify the objects in user interface such as web application elements and perform actions on the object. It provided the record and play tool that can generate script to ease the automation development. It also allows write VB lines to do some complicated test cases. However, QTP is costly to use and since it is not open source, and it is not possible to integrate with any robust framework like maven. Another drawback is that it only supports VB script and Windows operating system and limited browser. Additionally, the execution time of QTP is relatively high with high workload on CPU and RAM.

2.3.2 Watir

Watir is a free open source product based on Ruby, allowing the tester to manipulate objects such as HTML and JavaScript in a Web page by driving the Internet Explorer programmatically. The methodology behind is that Ruby has built in Object Linking and Embedding (OLE) capabilities, which allow different applications to modify a editing document so both the automation tool and internet Explorer can access to the web document at the same time. However, Watir is not a Record/Playback Tool thus a user should develop scripts to automate the test in Ruby language manually. And the browser support is only limited to Internet Explorer.

2.3.3 Selenium

Selenium automates browser by making calls to web browser and giving command to the browser web driver. It supports many development languages such as Java, ruby, python and almost all browsers such as IE, Firefox, Chrome, Safari, etc. Being into the open source world, Selenium can integrate with just about anything like maven and we can extend the functionalities of selenium by extending their source code based on this point. And testers can connect multiple nodes to run different tests in parallel. There are two components to be used in this project. Selenium IDE, as a Firefox extension, allows the tester to record and play back tests easily and fast on system under test and generate scripts at the same time. However, a record and replay tool such as Selenium IDE is designed to be a quick solution to automation, not a solution to a full regression test suite and, hence, could result in costly maintenance of the test suite. Selenium web driver employs browsers' native support like Google web driver to directly drive the browser. It is designed to support dynamic web pages where users interact with different views of the application without reloading the whole application. And one can also use Selenium web driver together with selenium grid and web sever so that the tests can be distributed over multiple machines even virtual one. It is designed to create robust, browser-based, scaled and distributed automation s across many environments. However, develop an automation test using Selenium web driver need expertise programming skills and requires expensive programmer time. Therefore, the combination of selenium IDE and selenium web driver could eliminate the disadvantages of each other.

2.3.4 Conclusion

As mentioned above, selenium supports most browsers and lots of programming languages, is an open source tool, can integrate with other framework and have lower execution time, which makes it best candidate among the existing automation testing tools. The main disadvantage of Selenium is the need of relative strong programming skill to develop a robust or scalable automation. Therefore, my approach is to integrate Selenium IDE and Selenium web driver, and reconstruct the code outputted by Selenium IDE to make the test automation script reusable and maintainable by using identified design patterns and separating data entry from the script.

2.4.1 Random testing and related works

In random testing, random inputs are passed to a web application, mainly to check whether the web application functions as expected and can handle invalid inputs. Random testing is easy to implement and automate with a simple concept. Random testing can also effectively detect failures especially in unexpected ways and it could be more general and adaptable for GUI testing since it does not rely on the source code and the specifications.

Pacheco & Ernst (2005) have first introduced the techniques to generate a pool of test inputs and select a small subset from the pool likely to reveal faults. Figure 1 depicts the components and the flow of their project. They employed a tool Daikon (Ernst et al. 2007) which can dynamically discover likely program invariants to construct an operational model of the software under test from proper executions of the software. An operational model contains `invariants` that hold at the entry and exit of the program's components. Candidate inputs are constructed with existing values and method calls derived from the operational model. They further classify each candidate as illegal (the input is illegal), normal operation, or fault-revealing (the input is fine, but the system has faults) by executing the candidate test inputs and comparing results against the operational model. They use guided approach selecting inputs labeled with normal operation into the candidate inputs pool to generate new input. The reducer further reduces the number of candidates by categorizing the fault-revealing candidates into various partitions and select one candidate from each partition. The oracle generation technique utilizes the operational model where the invariants can be achieved at runtime and can be used as test oracles. They have implemented a tool called Eclat to generate unit tests for Java classes. The input of Eclat is a set of Java classes and a correct program execution. The output of Eclat is JUnit test suite consisting of test inputs revealing possible faults and test assertions. However, this tool mainly works for unit tests and not for web applications and it relies heavily on the operation model constructed based on the existing execution.

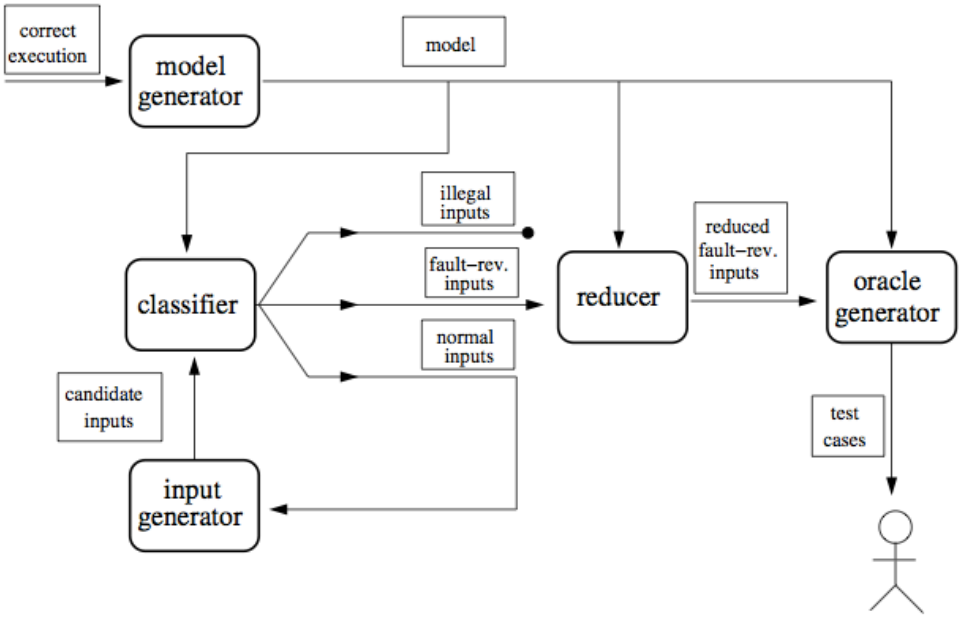


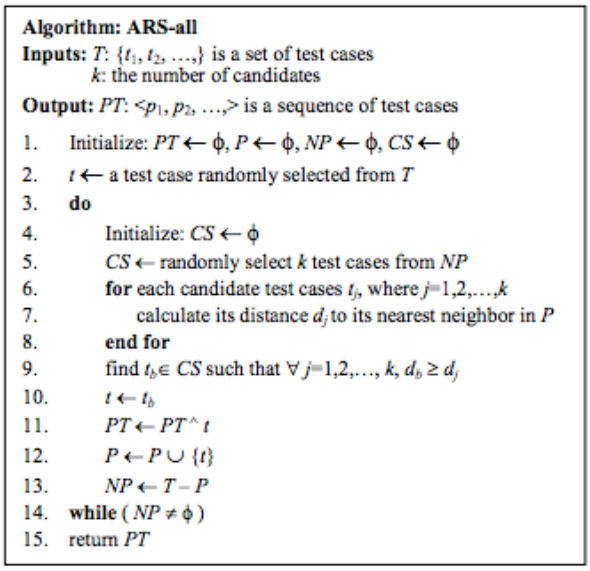
Figure 2 Components and the flow of Eclat (Pacheco & Ernst, 2005)

Afterwards, Pacheco et al. (2007) improve the random testing technique in a tool called Randoop. The test inputs are built based on the feedback from previously constructed inputs incrementally. As an input is created, it is executed and the execution result will be checked with the predefined contracts and filters. The input is identified as redundant, illegal, contract-violating, or useful for generating more inputs. The technique outputs an object-oriented unit-testing suite for the Java classes. Passing tests can be served as regression test suite; failing tests point to potential errors which should be corrected. An object-oriented unit test case is similar to object oriented selenium test case in the web application. A selenium test case is made up of a sequence of events and an assertion. They proposed an extension operation, which generates a new sequence by combining existing input sequences and method calls. And an extensible flag will be set to false if the value of the input is invalid and thus the search space is pruned.

Artzi et al.(2011) presented a framework called Artemis to automate the test case generation of JavaScript Web applications employing the feedback generating mechanism in the Randoop (Pacheco et al. 2007) project. Their idea is based on identifying and analyzing JavaScript code fragments and the event handlers. They demonstrate a motivating idea to collect relevant events by monitoring the execution. They illustrate an example that when function X reads a variable and Y writes that variable, it is plausible to test X again after Y to see whether the variable changes.

Another example is that if a function have several conditionals, it is likely that executing it multiple times will increase test coverage. They include these ideas in their prioritizing function to increase the coverage. The main contribution of the paper is a framework for JavaScript applications adopting the feedback directed testing algorithm. Compared with Randoop, Artemis targets at JavaScript web applications while Randoop targets at object-oriented APIs, test inputs of Artemis are sequences of method calls. A major difference is that Randoop generate test inputs based on a fixed collection of methods, while Artemis needs to discover relevant events during execution. However, the dynamic analysis performed at runtime is time-consuming while static analysis is highly related to the programing language used. And even for JavaScript applications, Artemis still has some limitations such that the JavaScript source compacted with no line breaks could not be analyzed using their approach. As a result, it is less efficient and general.

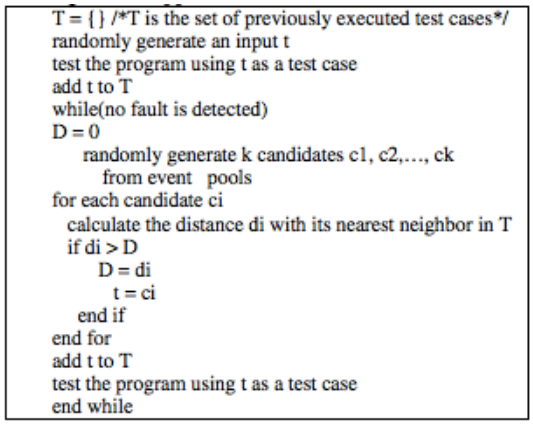
There are also many studies in other alternatives of directed random testing such as Adaptive Random Testing (ART). Chen et al. (2010) made a motivating observation based on many empirical studies (Ammann and Knight 1988; Bishop 1993) that many numerical program faults lead to contiguous failure regions of the program input domain, known as failure patterns. ART utilize these patterns and guides the randomly generating test cases. And Chen et al. (2010) presented an algorithm shown in Figure 3. The idea is to select a new test case from candidates based on their distance to the existing test cases. Inside the while loop, k test cases will be generated as candidates. And then the candidate test case with largest distance from nearest neighbor of existing test cases will be selected. Liu et al. (2010) identify that mobile application consists of user input events such as keyboard events and environmental context events such as GPS receiver. Therefore they adapt the ART algorithm (Figure 4) proposed by Chen et al in mobile application. However, they generate candidate test cases pool in a different approach. Firstly, they get the average length of event sequences from a user session or history profile. Then, they define different kinds of events. And for each kind, they generate an event pool for it. Further, they will randomly select an event from a random event pool to construct the sequence until the average length is achieved. The distance between test cases is the major part in ART algorithm, and Liu et al. (2010) define it as sequence distance and value distance.



Algorithm: ARS-all
Inputs: T: $\{t_1, t_2, \ldots,\}$ is a set of test cases
k: the number of candidates
Output: PT: $<p_1, p_2, \ldots,>$ is a sequence of test cases

1. Initialize: $PT \leftarrow \phi$, $P \leftarrow \phi$, $NP \leftarrow \phi$, $CS \leftarrow \phi$
2. $t \leftarrow$ a test case randomly selected from T
3. **do**
4. Initialize: $CS \leftarrow \phi$
5. $CS \leftarrow$ randomly select k test cases from NP
6. **for** each candidate test cases t_j, where j=1,2,…,k
7. calculate its distance d_j to its nearest neighbor in P
8. **end for**
9. find $t_b \in CS$ such that $\forall j$=1,2,…, k, $d_b \geq d_j$
10. $t \leftarrow t_b$
11. $PT \leftarrow PT^{\wedge} t$
12. $P \leftarrow P \cup \{t\}$
13. $NP \leftarrow T - P$
14. **while** ($NP \neq \phi$)
15. return PT

Figure 3 ART algorithm (Chen et al. 2010)



```
T = { } /*T is the set of previously executed test cases*/
randomly generate an input t
test the program using t as a test case
add t to T
while(no fault is detected)
D = 0
    randomly generate k candidates c1, c2,..., ck
      from event  pools
for each candidate ci
  calculate the distance di with its nearest neighbor in T
  if di > D
      D = di
        t = ci
   end if
end for
add t to T
test the program using t as a test case
end while
```

Figure 4 ART algorithm (Liu et al. 2010)

To conclude, the following table makes comparison of the above random testing technique and identifies which is adaptable to this project.

Paper Name	Testing Technique	Major Methodology	Input parameter	output	target
Eclat: Automatic Generation and Classification of Test Inputs	Random testing	1. Discover likely program invariants to construct an operational model of the software's operation 2. Candidate inputs classification 3. Test oracle generation from operation model invariants	A set of classes to test and an example program execution	JUnit test cases	Mainly Java Object-oriented classes
Feedback-directed Random Test Generation	Feedback directed random testing	1. Build inputs incrementally from empty set. An input is executed and verified by contracts and filters once created. 2. Extension operation by concatenating its input sequences and appending a method call at the end.	A set of classes to test	JUnit test cases	Mainly Java Object-Oriented classes
A Framework for Automated	Feedback directed random	Identify and analyze JavaScript code fragments and the event handlers	Web application	Test suite	JavaScript applications

Testing of JavaScript Web Applications	testing				
Adaptive Random Testing of Mobile Application	Adaptive Random Testing	Adaptive Random testing method The new model defining distance between test inputs	The application and user session	Test suite	General mobile applications, event driven applications

Table 1 Comparison of studies on feedback& adaptive random testing

2.3.2 Crawling testing technique and related work

CrawlJax, proposed by Mesbah el at.(2008), targets at crawling dynamic document object model (DOM) in Ajax-Based Web applications to infer a model of navigation paths and states. It is challenging to crawl Ajax applications since it could contain a single page with a single URL and it is nontrivial to detect all states at running time. They adopted the solution to open the Ajax application in an embedded browser and identify elements which are capable to trigger the state change based on tag name from root node and then use a controlled robot to simulate user actions where the DOM analyzer will check and produce a state-flow graph stored in Finite State Machine by recording the trails to the changes. A finite state machine usually has a finite set of states, inputs, outputs and transitions which determine the transition from the current state to a next state, depending on the input, and an output function, which determines the output produced by a transition. Nonetheless, there are some problems for this approach. Firstly, the method adopted to select elements which could trigger state change by identifying the element more likely exposed to an event type, is not feasible since every element in a page could be clickable depending on developers' knowledge and preference. Moreover, the way Mesbah et al. (2008) employed to go back to parent state is to save the state of the changed elements and to find the elements after reload. This is not accurate since there is no guarantee that they reach the exact same state after reload and you need to redo the steps every time a state changes. In addition, that the form elements requiring custom data are extracted at crawling time could break the crawling flow. Also, the sequence of events is hard to identify and thus the test coverage is not adequate.

2.2.3 Concolic testing technique and related work

Concolic testing techniques automate test input generation by combining the concrete and symbolic execution. The concrete part of concolic execution is where the program is normally executed with concrete inputs, drawn from Symbolic executions builds constraints and is often followed by a generation of concrete test inputs from these constraints. The goal in concolic testing is to generate different input data which would ensure that all paths of a sequential program of a given length are covered.

Saxena et al. (2010) developed a symbolic-execution based framework Kudzu for client-side JavaScript code analysis mainly focusing on finding client-code injection vulnerabilities. Kudzu classifies JavaScript input space into event spaces and values spaces. Event space consists of states and sequence of actions while values space refers to external entities such as data from user. They propose a constraint language and build a practical solver called Kaluza that supports the specification of Boolean, machine integer, and string constraints. Figure 5 illustrates architecture of Kudzu. There are five core components shaded in gray. The GUI explorer selects a random ordering of user events and executes them. Concrete inputs of an execution is recorded, and then symbolically executed by the dynamic symbolic interpreter. The path constraint extractor takes the result of symbolic execution and constructs constraints with the aim to exercise different execution paths of the JavaScript code. The constraint solver solves the constraint by finding satisfying assignments to variables, therefore generating new values to be used as inputs. Finally the input feedback system sends the new generated inputs back to the JavaScript program to drive new executions.

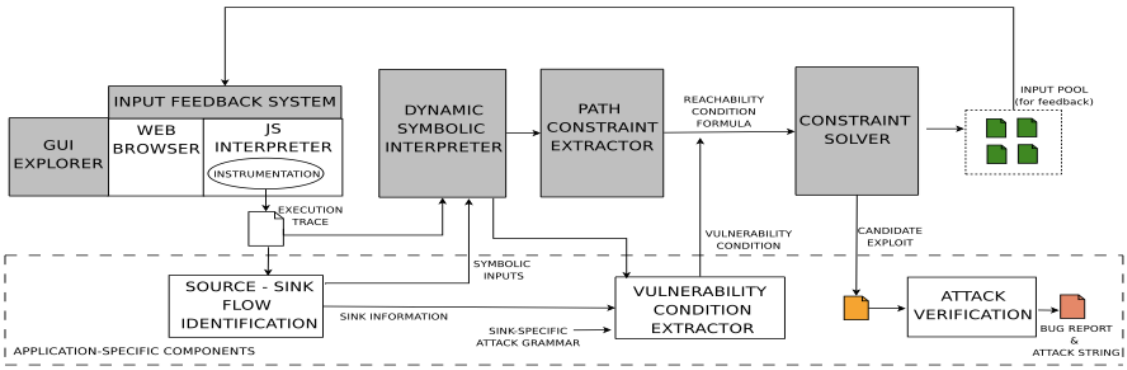


Figure 5 Overall architecture of Kudzu (Saxena et al. 2010)

2.2.4 Graph and Model based testing and related work

The graph and model based testing approach essentially creates a model of a web application. Test cases are then derived based on the model constructed. The test cases are generated according to either the all-statement or all-path coverage criteria. The graph and model based approach includes finite-state-machine-based testing, where a finite-state-machine depicting the model of the system is fist constructed, from which test cases are derived.

GUI Ripper is a reverse engineering tool used in GUITAR targeting at extracting a GUI model from GUI applications on Windows Operating System proposed by Memon et al. (2003). They propose to utilize event-flow graph to show all possible flow of different GUI events including user events and system events. GUI Ripper can dynamically explore, extract information about the application's GUI structure and perform the event extracted from available GUI widgets to construct the graph. Test cases can be further generated from the event-flow graph by traversing the paths. Moreover, there are many studies in GUI pattern and GUI test patterns from which some model-based testing techniques derive. GUIs consist of GUI patterns like form, data entry fields, authentication and so on, which derive corresponding test strategies (Cunha et al. 2010). Taking authentication pattern as an example, there are mainly three classes of strategies: correct user name with correct password, wrong user name and wrong password. There have been some studies on the GUI test pattern based on identified GUI patterns. Moreira et al. (2013) have proposed some GUI test patterns such as Input UI test pattern, Login UI test pattern, Sort/Find UI test pattern and so on. The generic testing solutions to GUI patterns leads to the advances of automated model based testing. MBT approaches are often employed in generating test cases automatically based on different model constructed (Cunha et al. 2010; Moreira et al. 2013). Cunha et al. (2010) present a pattern-based approach for automating GUI test and implement it in a tool called PETTool. Figure 6 illustrates architectural structure of PETTool. It identifies patterns in GUI and generates generic solutions for each pattern. The model categorizes basic patterns into structural and behavioral pattern. Controls like text boxes could be used by pattern. And structural pattern could consist of several behavioral patterns. For example, a login form is composed of data entry field and authentication. And each behavioral pattern has expected behavior, which determines the test result.

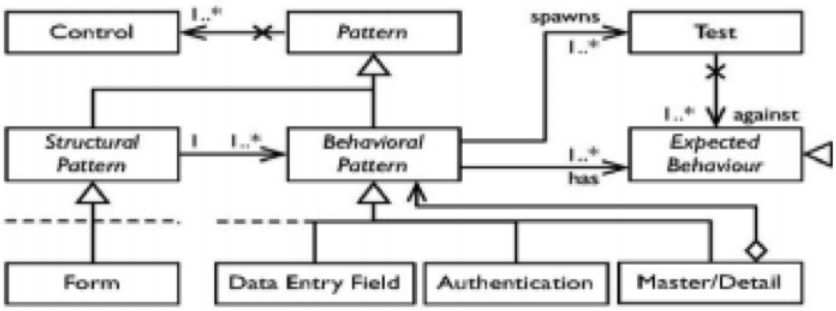


Figure 6 Architectural Structure of PETTool (Cunha et al. 2010)

Chapter 3 REQUIREMENTS AND DESIGN

3.1 Software requirements

3.1.1 Product perspective

This system will base on Selenium IDE and add advanced functions to Selenium IDE. The major objective of this system is to reduce the effort of constructing a well designed automated testing suite.

This system will call Selenium API which communicates with browser driver which in turn drive the corresponding browser to perform actions like ‘click’, ’type’ on the Application Under Test (AUT) which consists of Document Object Model (DOM), see Figure 7.

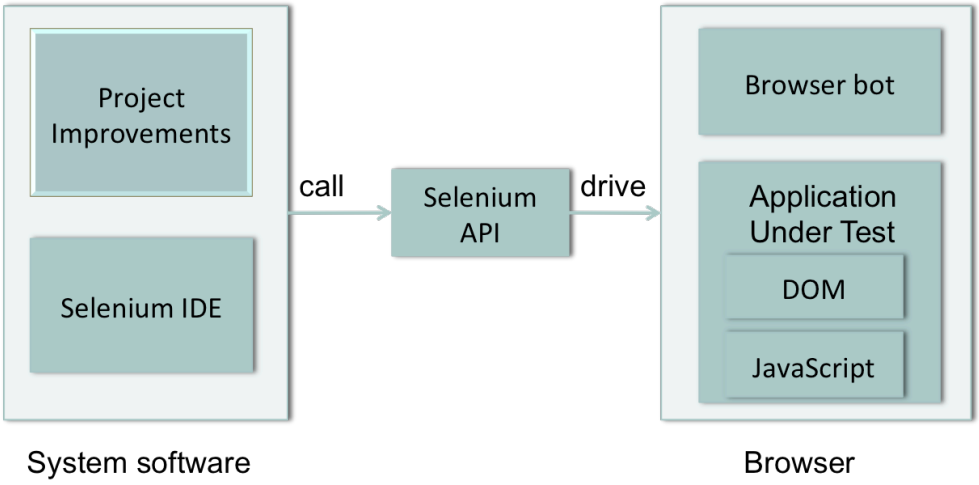


Figure 7 System from product perspective

3.1.2 Product functions

With this system, users will be able to construct a test suite script in Java without writing actual code with page object pattern applied.

3.1.3 User perspective

There is mainly one type of user – tester that interact with the system. Although this system could be extended to do respective administration work, it is not included in this version.

3.1.4 Functional Requirements (FRs)

3.1.4.1 Extract data variables

ID:	FR1
Title:	Extract data variables
Description:	After a user performed a sequence of actions, the UI elements and the input data will be extracted and stored as variables and displayed in data variable view. User can edit the variables in data variable view. The new added value variable can be reused displayed in the auto-completion list when manually editing a command value.
Dependency:	None

Table 2 FR1 - Extract data variables

3.1.4.2 Add page objects

ID:	FR2
Title:	Add page objects
Description:	User can define the Page-objects structure of the AUT and can add the page objects in page object view. As we mentioned, Page-object pattern is to provide abstraction between test cases and the AUT utilize a single view or page as an object which have properties and function.
Dependency:	None

Table 3 FR2 - Add page objects

3.1.4.3 Group commands as a function

ID:	FR3
Title:	Group commands as a function
Description:	Given there are repetitive steps to record, a user can select those commands and group these commands into a function belong to a page. This function can be reused and called directly afterwards across test cases. In doing so, we can abstract out the execution details for the function such that if they were to change at some point, you would have a single point of update. We can save the function belonging to a page object. And the function will be added into the action auto-completion list to ease the reuse of the function.
Dependency:	FR2

Table 4 FR3 - Group commands as a function

3.1.4.4 Generate random data for input from a data pool

ID:	FR4
Title:	Generate random data for input from a data pool
Description:	When a user is testing form or input box, the system will extract a list of input elements from current page and randomly pick up the test data from data pool according to the data type of the input element. The system will be able to analyze the data type of the input element according to the value of the attributes such id, name. If the system provide the wrong data type, the user is able to select the correct data type. And the generated data could be filled into the input element automatically.
Dependency:	None

Table 5 FR4 - Generate random data

3.1.4.5 Get data from user import data

ID:	FR5
Title:	Get data from user import data
Description:	User can import data and the data will be stored as value variable. When a user is testing form or input box and after he click 'get user data' button, the system will extract a list of input elements from current page and provide a selection box of the stored data for selection. The system will be able to analyze the data type of the input element according to the value of the attributes such id, name. If the system provide the wrong data type, the user is able to select the correct data type. And the data could be filled into the input element automatically.
Dependency:	None

Table 6 FR5 - Get data from user import data

3.1.4.6 Convert low-level script to well designed test suite

ID:	FR6
Title:	Convert low-level script to well designed test suite. Output the test suite in Java format with page object pattern and with UI elements variables and value variables in separate property files.
Description:	When a user finished a test suite, he can output the test suite in page

	object pattern in java. The output test suite will have well designed file structure with the UI elements variables and value variables converted into a separate UI map file and value map file. The code inside the single files will be reconstructed and the same logic like login will be removed to the login function under login page object. When there is a change in UI elements or login logic, there will be only one point change. This could save considerable maintenance effort as the application scale up.
Dependency:	FR1, FR2, FR3

Table 7 FR6 – Convert low-level script

3.1.4.8 Generate valid test data sets randomly

ID:	FR7
Title:	Generate valid test data sets randomly
Description:	After a user generates a set of data, he is able to generate more sets of valid data according to the existing data type randomly. These data sets will be tested once generated.
Dependency:	FR4

Table 8 FR7 – Generate valid test data

3.1.4.9 Generate test cases randomly

ID:	FR8
Title:	Generate test cases randomly
Description:	After a user generates a set of random data, he is able to generate more test cases according to this test case randomly to increase the test coverage.
Dependency:	None

Table 9 FR8 – Generate test cases randomly

3.2 Design

3.2.1 System architecture design

My project is to make the test automation development process for web application easier. It is based on the record and replay tool selenium IDE which can record the tester action and generate low level scripts which enable the IDE replay the actions.

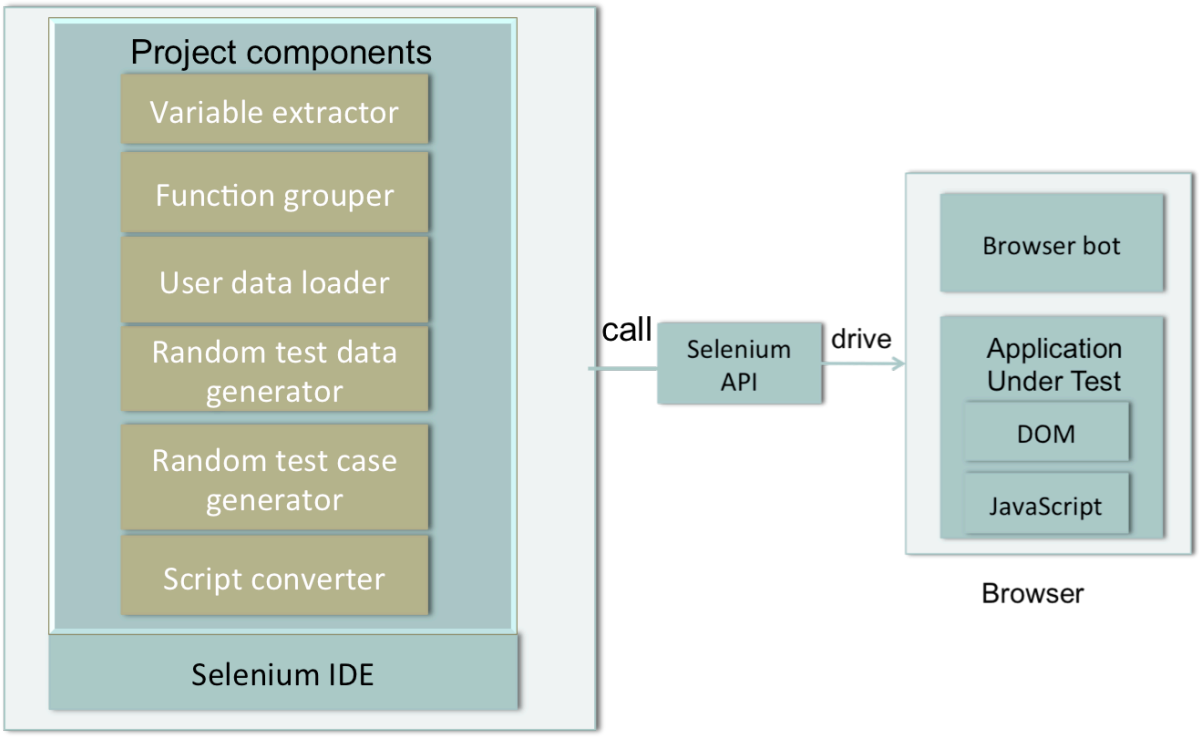


Figure 8 System Architecture design

The project will contains the following high level components (Figure 8):

3.2.1.1 Variable extractor

Variable extractor can extract the UI elements and the input data and store them as variables.

3.2.1.2 Functional grouper

Tester can select some repetitive commands and group these commands into a function belong to a page via Functional grouper.

3.2.1.3 Script converter

Script convertor is to refactor the test script (low level script output by selenium IDE) of an individual test case and put all test cases in structured and encapsulated test suite currently adopting page object pattern. The script converter will replace the hardcoded user input data and UI elements in the low level scripts produced by selenium IDE.

3.2.1.4 User data loader

This component can load user-defined data or previous generated data for form elements with some user interventions.

3.2.1.5 Random test data generator

This component can generate random data for form elements or input with some user interventions.

3.2.1.6 Random test case generator

The test case generator will also generate some test cases based on existing test cases to further more effort.

3.2.2 Class diagram and class design

Selenium IDE records every user-action as a command consisting of action, target and value and treats a test case as a sequence of commands. A Function object is a group of commands and a Page-object consists of functions manipulating the DOM of that page. Moreover, a test suite also contains one or several test cases.

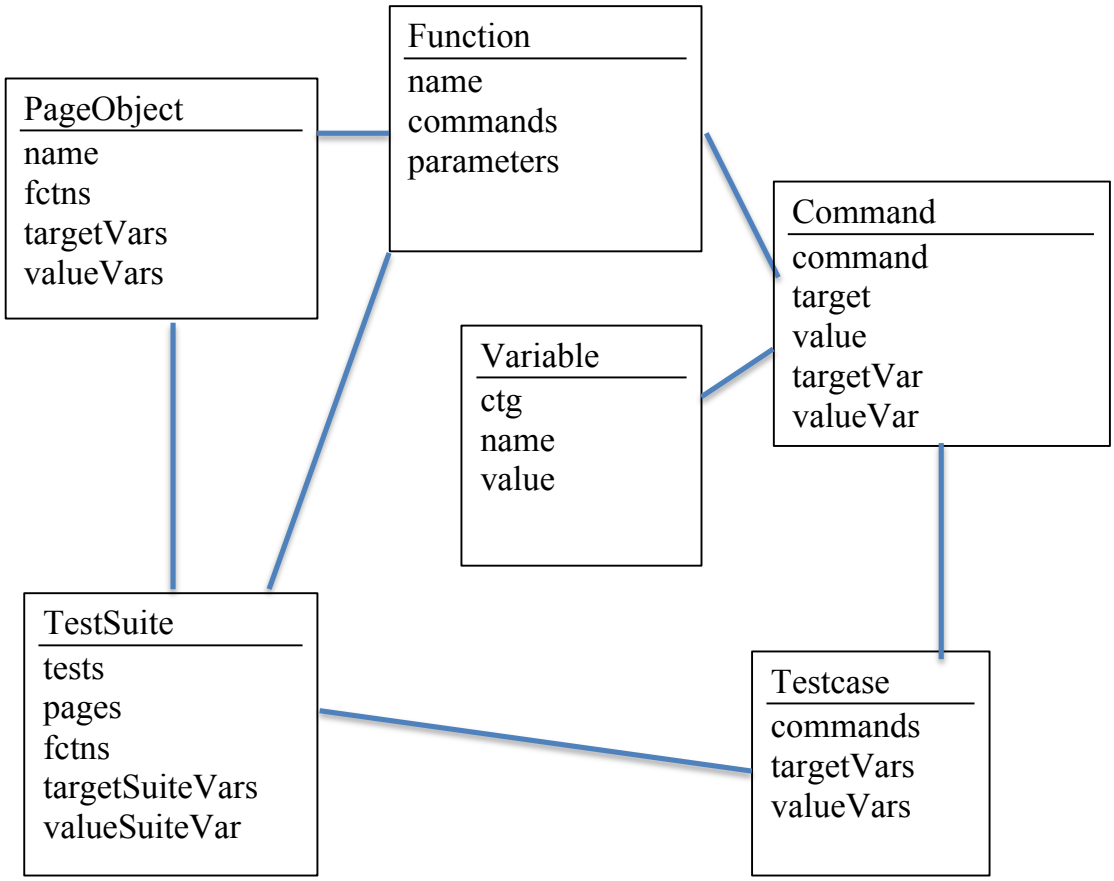


Figure 9 Class diagram

Followings are the detail and sample of the class objects:

Command object:

Figure 10 Command object

Test case object:

Figure 11 Test case object

Test suite object:

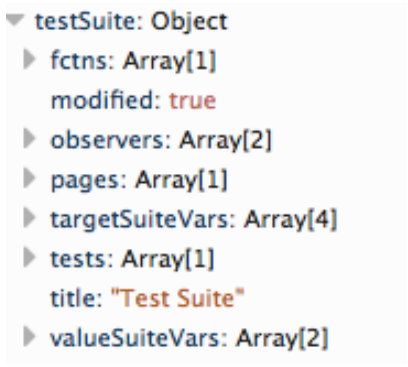


Figure 12 Test suite object

Function object:

Figure 13 Function object

Page object:

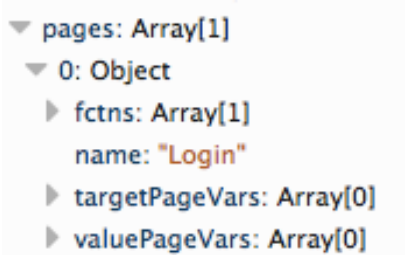


Figure 14 Page object

Value variable object:

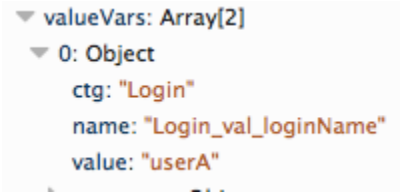


Figure 15 Value variables object

Target variable object:

Figure 16 Target variables object

3.2.2 Component and detailed design

3.2.2.1 Design pattern and technique used

3.2.2.1.1 Adaptor pattern

Adaptor pattern provide an interface for two classes which could be incompatible to each other. In this project, we implement a format adaptor for the script convertor to call functions in Java formatter in order to output the script in Java format. This project will only cover Java formatter. So we also implement multiple inheritances, which allow the application extend to support different languages such as Python, Ruby in future.

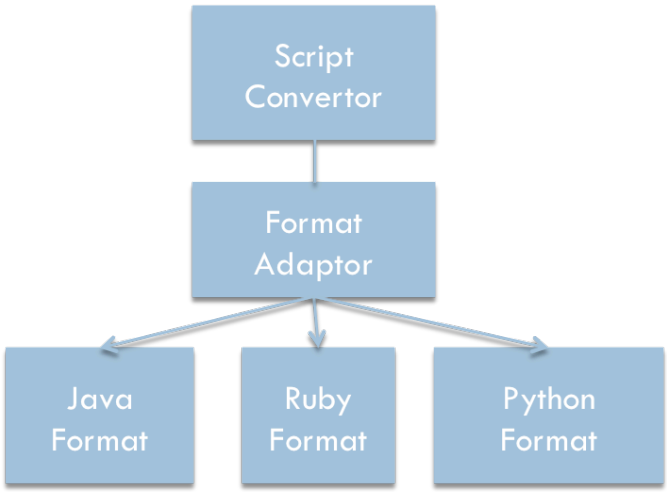


Figure 17 Adaptor format for formatting script

3.2.2.1.2 Prototyping technique

When in the designing stage of this project, prototyping technique is used to demonstrate the basic concepts of the system and provides a clear interface for user to have a basic idea of the behavior of the system.

3.2.2.2 User flow and algorithm design for functional components

3.2.3.1 Extract data variables

Selenium API contains a browser-bot object which speaks to and controls the browser. The browser-bot can get the documents and monitor the page load of the web application. In this project, we would employ the browser-bot to get the attribute and value of current interacting elements and page title during capturing. We will combine the page title and UI attribute value and form an element name. If the name contains invalid characters, we will replace it with '_'. UI elements and Value elements variables will be displayed in the variable views and user can view the variable name and edit it.

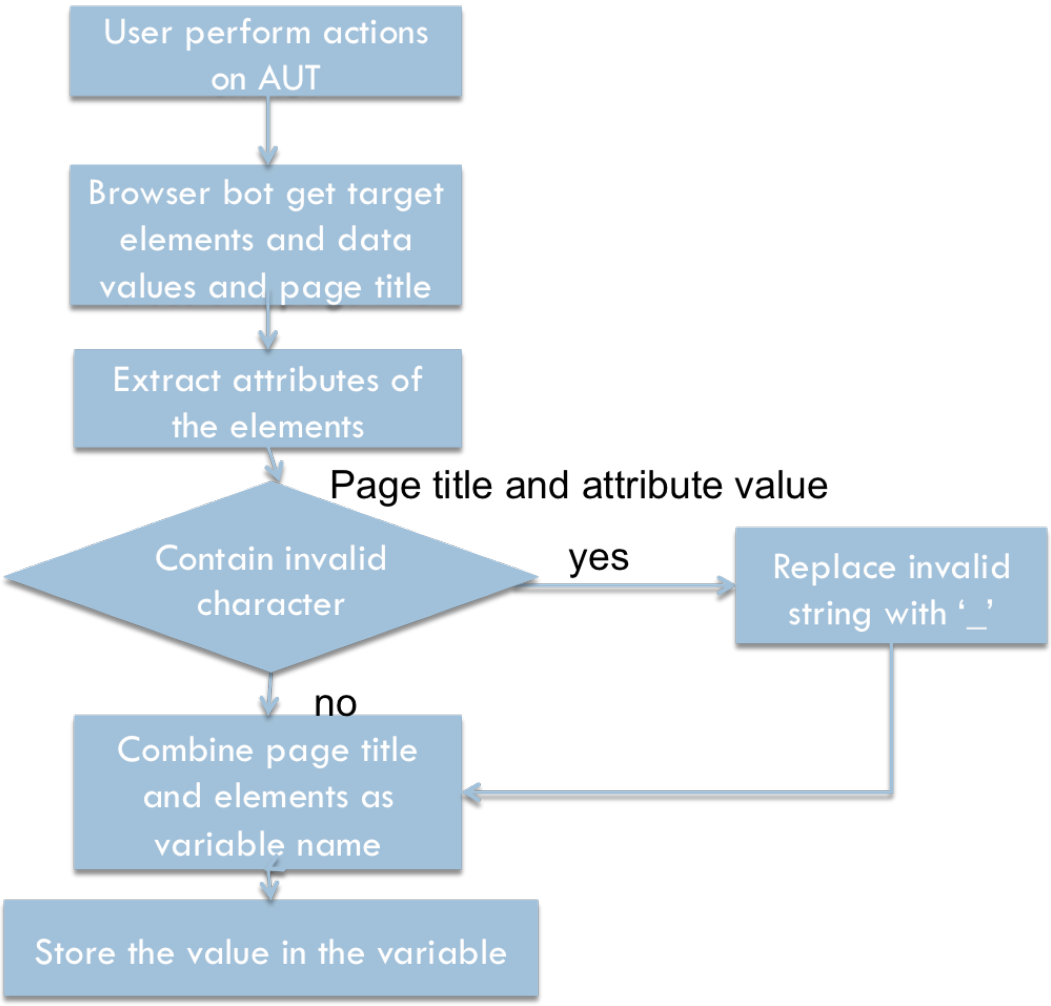


Figure 18 Extracting data variables algorithm

3.2.3.2 Page object

Tester can add page object in Page objects view. A page object does not necessary to be a whole page.

1. User design a number of pages for the AUT
2. User add page objects in Page object view
3. The Page object will be pushed into to the array of pages of the test suite

3.2.3.3 Group commands as function and reuse function

Tester can select several commands and group as a function belong to a page object and this function can be reused and called directly afterwards across test cases. We can save the function to a page object.

3.2.3.3.1 Group commands:

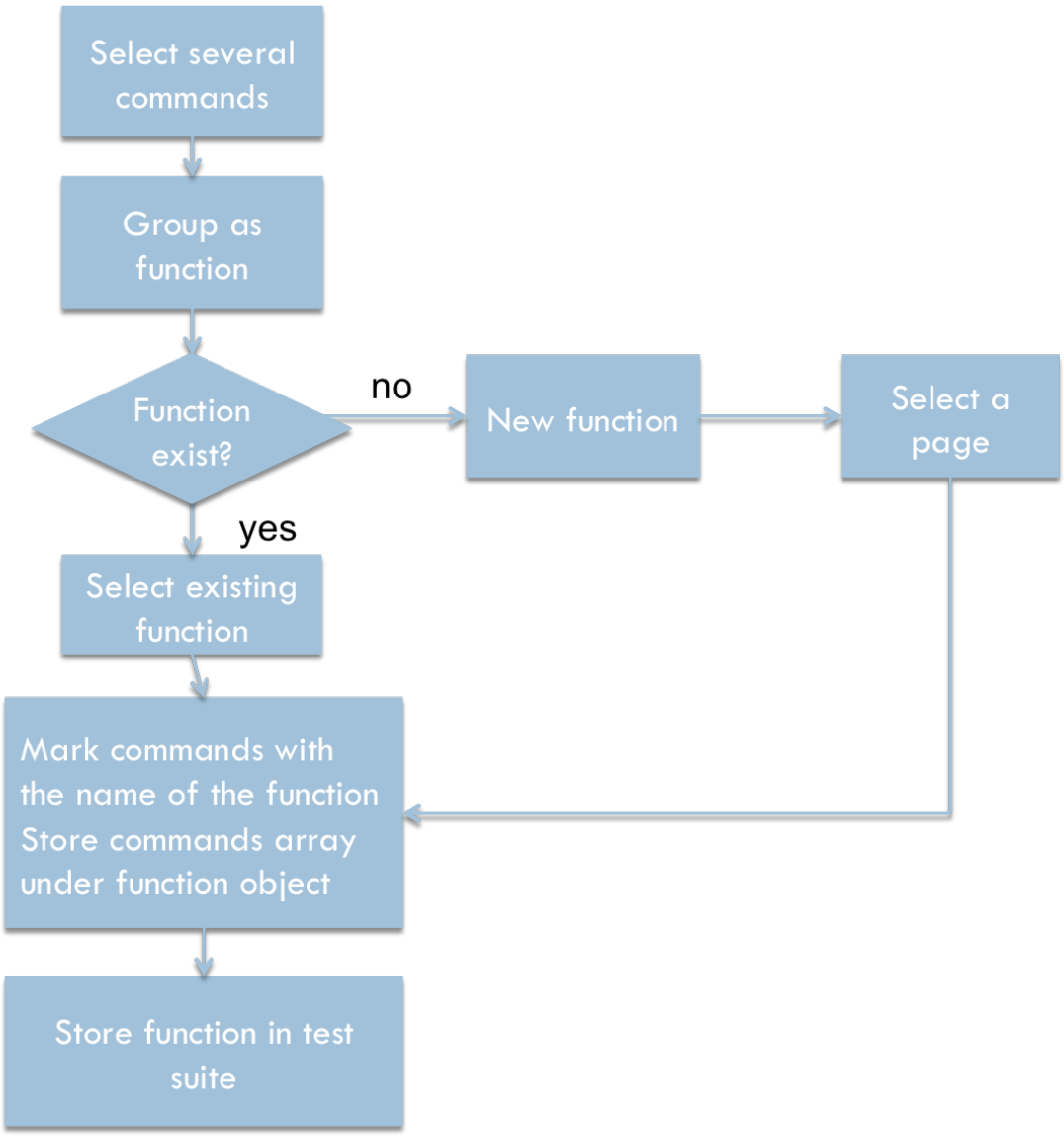


Figure 19 Grouping commands algorithm

Reuse commands:

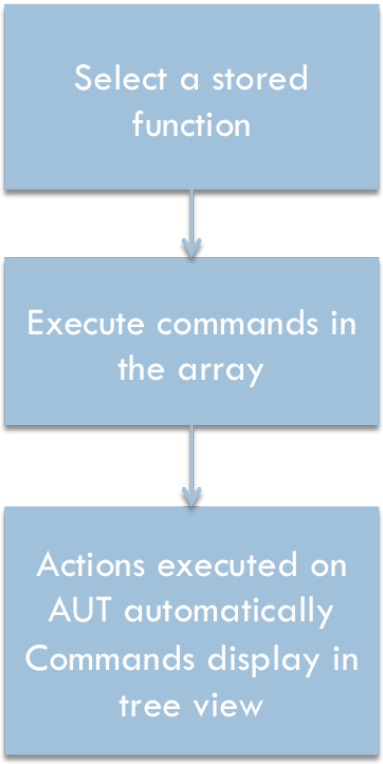


Figure 20 Reuse commands algorithm

3.2.3.4 Random test data generator

When the tester is testing the elements, test data generation could help generate a set of data according to the input data type. The system utilizes the predefined data pool provided by Chance.js. Chance.js is an open source random data generator licensed under MIT license. The system crawler will extract the input box information and the system will analyze the value of input text box attribute such as name, id and adopting Levenshtein distance (Navarro 2003) to find the most similar string and then decide the data type fitting the text box. Levenshtein distance measure the minimum single character edit to change one character to another. This methodology highly depends on the input attributes naming the web application developer. However, the system provide the selection box and tester can select the correct data type or edit the value if the resulting data type is not correct.

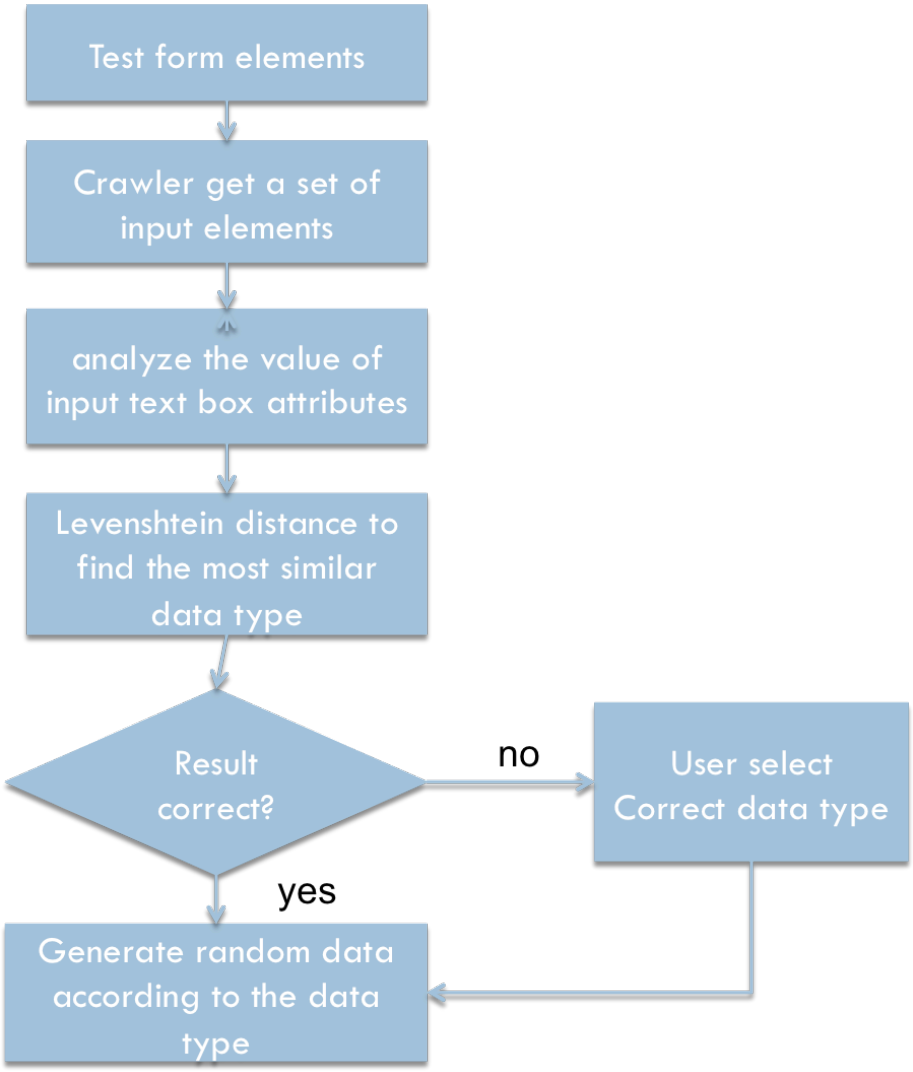


Figure 21 Random test data generation algorithm

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 22 Levenshtein distance (Navarro 2003)

3.2.3.5 Get user data

Tester can import his own data or previous generated data file into the system. And these data will be stored as value variables for use. The system will read the data file using the Firefox file and directory service API and use regular expression to get the variable name and variable value. And the remaining part will be similar as random test generation adopting Levenshtein distance to find the appropriate data name.

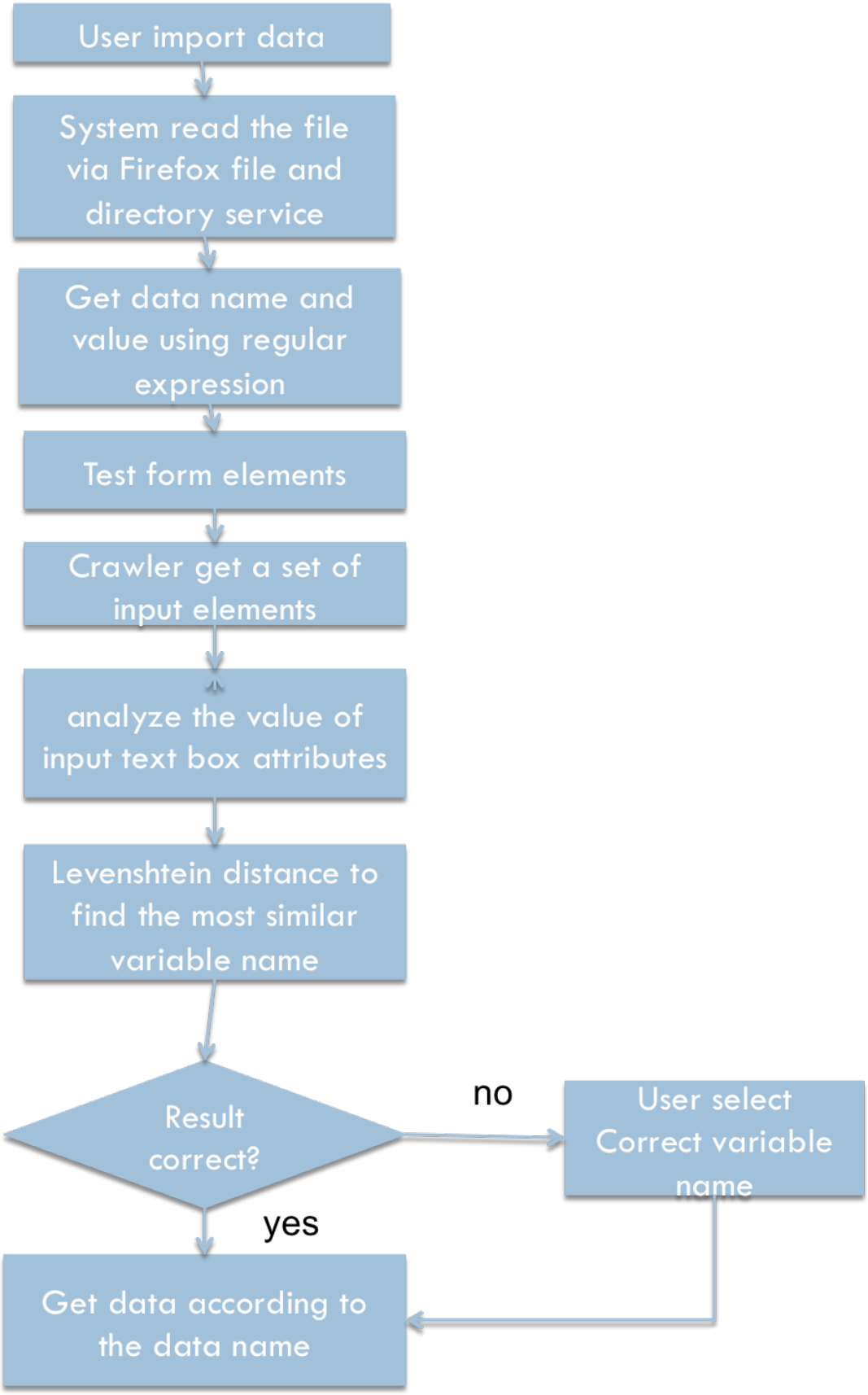


Figure 23 Get user test data algorithm

3.2.3.6 Script converter

Script convertor is to refactor the test script (low level script output by selenium IDE) of an individual test case and put all test cases in structured and encapsulated test suite currently adopting page object pattern. The UI elements variables and user input data will be output to a resource file within the test suite. With the original Selenium IDE, test cases are independent to each other make the function not reusable and user input data and UI elements are scaled across test cases. After processed by converter and applied Page Object Pattern, the code inside every single file will be reconstructed and the same logic like login is removed to the function under login page object. This could save considerable maintenance effort as the application scale up.

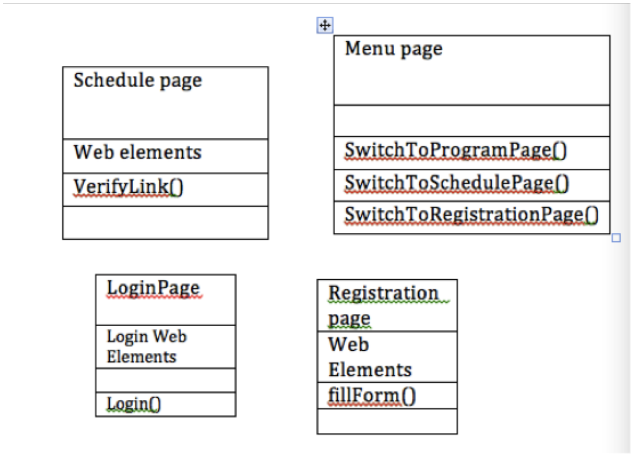


Figure 24 Page-object pattern

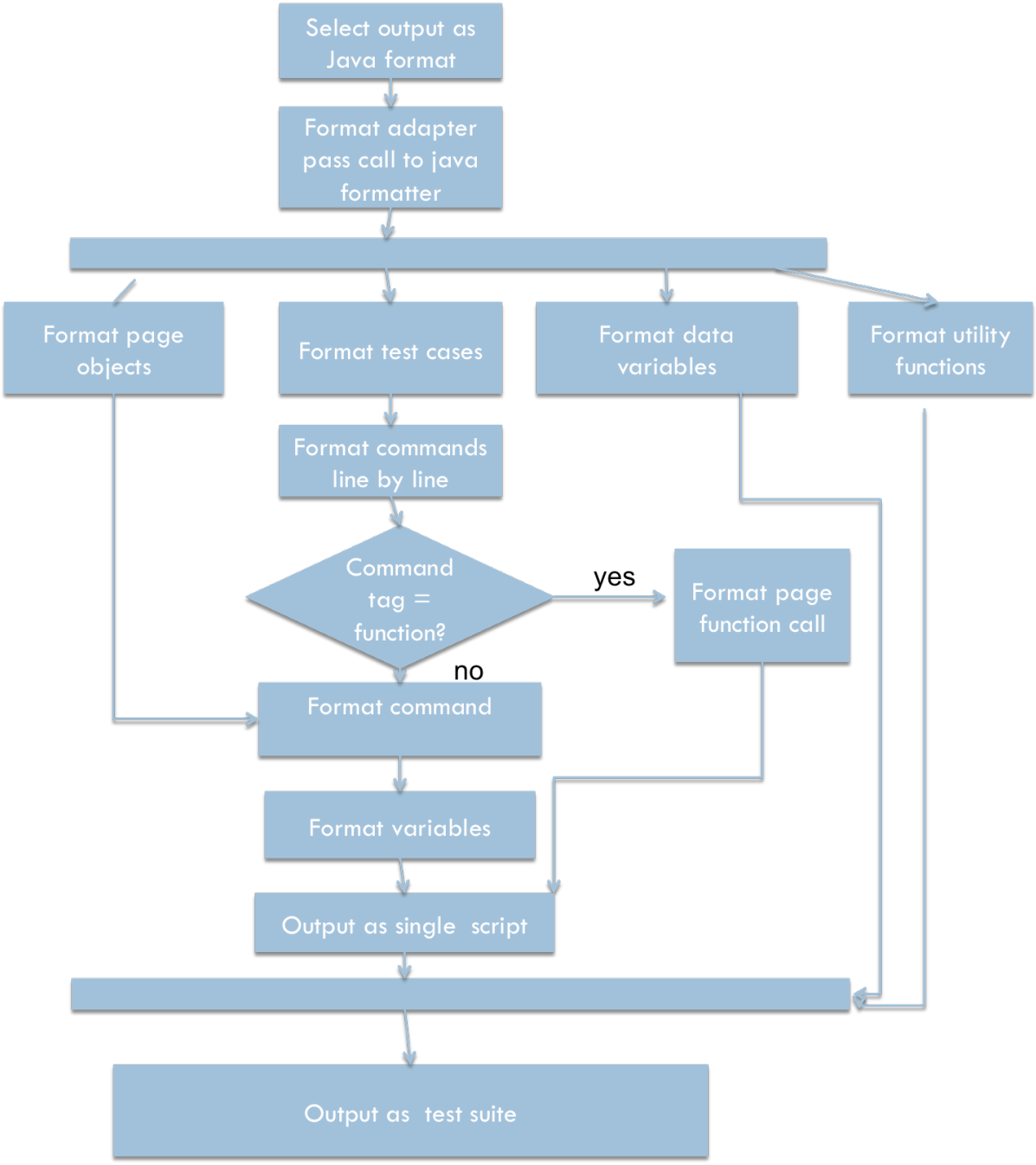


Figure 25 Script converter algorithms

3.2.3.7 Random test case generator

The algorithm to generate test case is shown in Figure 26. Compared to the ART algorithm in Chen et al. (2010), we propose a different approach to generate candidate test data. We will adopt chance.js to generate some random string. And then select an existing test case and choose one of its test value to modify. We will select the most faraway data from this chosen test value by comparing the distance of this two value adopting the normalized levenshtein_distance. And our aim is to find a candidate test case Cj generated from an existing test case Ti which is most faraway from this Ti among candidates input to improve the coverage. As motivated by ART algorithm, the

test cases more faraway from passed test cases are more likely to reveal defects. It first initializes an empty set T to store the generated test suite. And the worklist is the existing test suite which we will derive new test cases from. Every time we will select a test case from the working list and new generated test case will be also added to the working list. Therefore, we also need to specify a threshold to stop the loop otherwise the loop could be infinite. And the test case will be tested against the AUT once generated if the test case passed, it will be added to T and the Worklist.

```
Algorithm generateTestcases(TestCase, k,threshold)

T = {TestCase}
Worklist = TestCase.valuesVars
While(!threshold)
v = worklist.Next();
t =  clone(TestCase)
D=0;
randomly generate k candidates c1,c2......,ck
for each candidate cj
    calculate the distance cj with v.value
 if di>D
    d=Dj v=cj t.cj=cj
 end if
  test the program using t as a TestCase
  add ti to T
  add ti to Worklist
end while
```

Figure 26 Algorithm pseudo-code

We further define the distance between the generated test data value with existing test case data value using modified levenshtein_distance. We first define several categories of the data such as number, a-z, A-Z, '@','._\'','=<>+' assign 1,2,3,4,5.6 to these 6 categories as we believe that test data A1ggg is very close to B3fff. Therefore

Number	1
a-z	2
A-Z	3
@	4
.\'	5
=<>+	6

Table 10 Character type matching

Expected sample results:

Test Data	Expected Result
A1ggg	31222
B3fff	31222

Table 11 Sample result of normalizing character type

3.2.4 User interface design

3.2.4.1 Main User Interface design

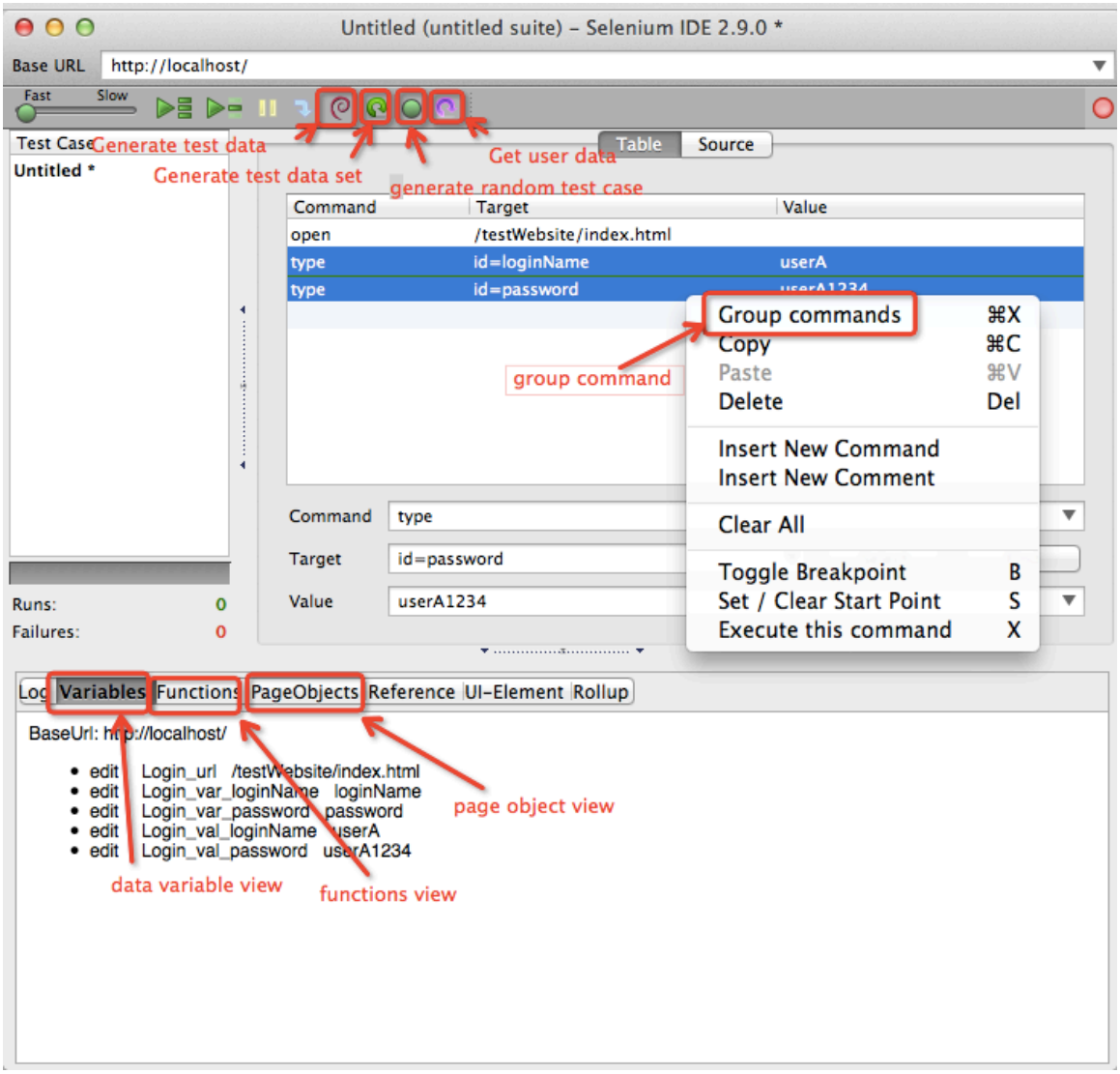


Figure 27 Main User interface

User interface is based on the Selenium IDE (Figure 27). The UI elements circled by red rectangle are added into the User Interface of Selenium IDE to perform the corresponding functions. Selenium IDE is an extension of Firefox using XUL language which is similar to HTML and XML together with overlay.js (JavaScript API). Selenium IDE has a left panel displaying a list of test cases you created. And on the right panel which is the test case editor table of current editing test case. Each row

stands for one test step. And one step contains command, target and value. Selenium IDE has a list of pre-defined command like click, clickandwait and type. Target is the UI element you are performing action. And value is the parameter you passed to the action. For example, that type “gmail” in the textbox will be understood as:

command	target	value
type	textbox	gmail

Table 12

System Crawler will get the value and properties of UI elements. These elements will be processed to be stored as variable and list in the DataVariables view in the bottom of the interface. The table is editable and the tester can edit the variable name as appropriate. Moreover, you can select multiple test steps, right click and select group as commands to add a function. The resulting function can be seen in function view in the bottom of the interface. Tester can select an existing object or create a new object which the function belongs to. This table is also supposed to be editable. These saved objects and functions can be reused in other test case construction. In later stage, it is proposed to identify the repeated sequences without manual intervention.

Chapter 4 IMPLEMENTATION

4.1. Development environment and tools

4.1.1. Hardware configurations

Figure 28

4.1.2 Software configurations

Following softwares have been used in the Firefox extension development.

1. Komodo Edit 8
2. Eclipse Standard/SDK Version: Kepler Service Release 2 Build id: 20140224-0627
3. Firefox latest
4. Selenium IDE 2.9.0
5. Selenium-2.45.0
6. DOM Inspector, Error console, Browser console

Table 13 software configurations

Komodo Edit is especially good for Firefox extension development. And eclipse is used when trying to edit the output script by the system. Different from normally web application, it is not able to debug Firefox extension using Firebug and inspectors.

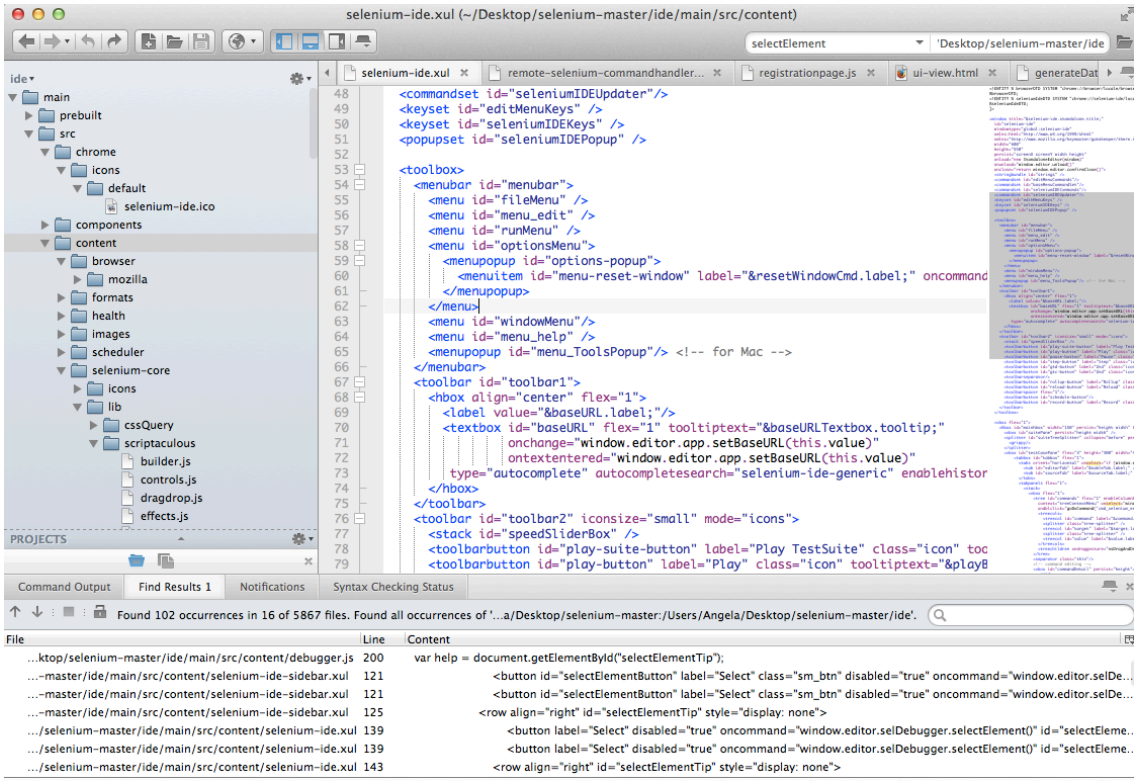


Figure 29 Komodo Edit editor

DOM Inspector is a tool that can be used to inspect and edit the live DOM of any web document or XUL application. The DOM can be navigated using a two-paned window displaying a variety of different views on the document and all nodes within.

4.2 Detail implementation of functional components

4.2.1 Data Variables

4.2.1.1 Extract target variable name of UI elements

The browser bot will get the target candidates by id, name or xpath and the target will be tested using regular expression on whether contains illegal character for variable name in programming language such as space which will be replaced by '_'.

```
function processTargetVar(targetCandidates){
        if (targetCandidates instanceof Array) {
                var res = targetCandidates[0][0].split('=');
                var result = res[1];
                if (/[\W]+/g.test(result)) {
                        result = result.replace(/[\W]+/g,"_");
                }
                return "var_"+result;

        }else{
                var res = targetCandidates.split('=');
                var result = res[1];
                if (/[\W]+/g.test(result)) {
                        result = result.replace(/[\W]+/g,"_");
                }
                return "var_"+ result;

        }
```

Figure 30 Process target variable

The same logic applied to extracting value variable name of user input data

4.2.1.2 Autocomplete for the reuse of existing value data

```
Editor.GENERIC_AUTOCOMPLETE = Components.classes["@mozilla.org/autocomplete/search;1?name=selenium-ide-generic"].
                              getService(Components.interfaces.nsISeleniumIDEGenericAutoCompleteSearch);
```

```
        loadCommandValueVars: function(){
                var valueBox = this.document.getElementById("commandValue");
                valueBox.setAttribute("enablehistory", "true");
                valueBox.disableAutoComplete = false;
                if (this.editor.app.getTestSuite().valueSuiteVars) {
                        //code
                var valueVars = this.editor.app.getTestSuite().valueSuiteVars;
                var locators = [valueVars.length];
                for (var i = 0; i < valueVars.length; i++) {
                    locators[i] = "$"+valueVars[i].name;
                }
                Editor.GENERIC_AUTOCOMPLETE.setCandidates(XulUtils.toXPCOMString(
                                                this.editor.getAutoCompleteSearchParam("commandValue")),
                                                        XulUtils.toXPCOMArray(locators));
                }
        },
```

Figure 31 Load command value variables into auto completion

We employ the autocomplete API provided by selenium IDE and put the value variables into the autocomplete pool. And the value variables will be add a prefix ”$” to ease later processing.

4.2.2 Group commands as function and reuse function

4.2.2.1 Group commands

Script to open ‘group as function’ dialog, the dialog is written by XUL since it is also a extension object.

```
        editFunction: function() {
            var testSuite = this.editor.app.getTestSuite();
            var self = this;
            if (testSuite) {
                window.openDialog('chrome://selenium-ide/content/create-function-dialog.xul', 'function-dialog', 'chrome,modal',
                                  testSuite, function(fname,pname,fexist) {
                        self.groupCommands(fname,pname,fexist);
                    });
            }
        },
```

Figure 32 Group as function

Get the range of the select commands and push it into the editing pool

```
499         groupCommands: function(fname,pname,fexist) {
500         if (fname == "")
501                 return;
502             var count = this.selection.getRangeCount();
503             if (count > 0) {
504                 var groupRanges = [];
505                 var currentIndex = this.tree.currentIndex;
506                 for (var i = 0; i < count; i++) {
507                     var start = new Object();
508                     var end = new Object();
509                     this.selection.getRangeAt(i, start, end);
510                     var groupCommands = {start: start.value, commands:[]};
511                     for (var v = start.value; v <= end.value; v++) {
512                         var command = this.getCommand(v);
513                         if (command != this.newCommand) {
514                             groupCommands.commands.push(command);
515                         }
516                     }
517                     groupRanges.push(groupCommands);
518                 }
519                 this.executeAction(new TreeView.GroupCommandsAction(this, groupRanges,fname,pname,fexist));
520             }
521         },
```

Figure 33 Group as function

Set the ‘ctg’ attribute to ‘fctn’ and set function name to the commands grouped into the function. And store the commands array under the function object. If the function not exists, add it to the page object and add the data variables to the page object.

```
935 TreeView.GroupCommandsAction.prototype = {
936         execute: function() {
937                 var currentIndex = this.treeView.tree.currentIndex;
938                 for (var i = this.ranges.length - 1; i >= 0; i--) {
939                         var range = this.ranges[i];
940                         for(var j =0; j <  range.commands.length;j++){
941                                 if (!this.fexist){
942                                         range.commands[j].setFname(this.fname);
943                                         range.commands[j].ctg = 'fctn';
944                                         range.commands[j].fctn = this.fctn;
945                                         this.fctn.commands.push(range.commands[j]);
946                                 }else{
947                                         range.commands[j].setFname(this.fctn.name);
948                                         range.commands[j].ctg = 'fctn';
949                                         range.commands[j].fctn = this.fctn;
950                                 }
951                         }
952                 }
953                 if (!this.fexist){
954                 this.fctn.parameters = getValueVarList(this.fctn.commands);// could this be value vars
955                 this.fctn.targetFctnVars = getTargetVarList(this.fctn.commands);
956                 this.fctn.page = this.page;
957                 //Todo use join
958                 mergeVars(this.page.targetPageVars, this.fctn.targetFctnVars);
959                 mergeVars(this.page.valuePageVars,this.fctn.parameters);
960                 this.treeView.reloadSeleniumCommands();
961                 }
962
963         },
```

Figure 34 Group as function

4.2.2.2 Call the stored function

Call and execute stored function after select the function from command list. Firstly get the commands array from the function and paste the command into the table and execute the command.

```
479         expandFctn: function(fctnName){
480                 if (this.editor.app.getTestSuite()) {
481                         var currentIndex = this.tree.currentIndex;
482                         for(var i=0;i<this.editor.app.getTestSuite().fctns.length;i++){
483                                 if (fctnName==this.editor.app.getTestSuite().fctns[i].name) {
484                                 var fctn = this.editor.app.getTestSuite().fctns[i];
485                                 this.executeAction(new TreeView.PasteCommandAction(this,currentIndex,fctn.commands));
486                                 var that = this;
487                                 for(var j=currentIndex;j<this.editor.app.getTestCase().commands.length;j++){
488                                  this.editor.selDebugger.executeCommand(this.editor.app.getTestCase().commands[j]);
489                                 }
490                                 }
491                         }
492                 }
493         },
```

Figure 35 Call the stored function

4.2.3 Random test data generation

We employed minimum Levenshtein distance to find the most similar string. Firstly we manipulate the JSON object of the data types and store all data type name in an array. We compare the Levenshtein distance of the valid attribute value of a input box such as name/id with every data type name in the array and find the minimum one.

```
var DataTypeList = {
    "Basics":[
        "string",
        "bool",
        "character",
        "floating",
        "integer",
        "natural",
        "password"
    ],
    "Text":[
        "paragraph",
        "sentence",
        "syllable",
        "word"]

    ,
    "Person":[
        "age",
        "birthday",
        "cpf",
        "first",
        "gender",
        "last",
        "name",
        "prefix",
        "ssn",
        "suffix"],

    "Mobile":[
        "android_id",
        "apple_token",
        "bb_pin",
        "wp7_anid",
        "wp8_anid2"],

    "Web":[
```

Figure 36 part of data type provided by chance.js

The levenshtein_distance algorithm, adapted from java in Wiki to JavaScript:

```
function levenshtein_distance (a, b) {
  if(a.length == 0) return b.length;
  if(b.length == 0) return a.length;

    //initialize the matrix
  var resultDP = [];
  for(var i = 0; i <= b.length; i++){
    resultDP[i] = [i];
  }
  for(var j = 0; j <= a.length; j++){
    resultDP[0][j] = j;
  }


  for(i = 1; i <= b.length; i++){
    for(j = 1; j <= a.length; j++){
      if(b.charAt(i-1) == a.charAt(j-1)){
        resultDP[i][j] = resultDP[i-1][j-1];
      } else {
        resultDP[i][j] = Math.min(resultDP[i-1][j-1] + 1,
                                  Math.min(resultDP[i][j-1] + 1,
                                           resultDP[i-1][j] + 1));
      }
    }
  }

  return resultDP[b.length][a.length];
}
```

Figure 37 levenshtein_distance algorithm

```
function findMinimumLeDistanceInArr(a,arr){
    var minD = levenshtein_distance(a,arr[0]);
    var result = arr[0];
    for(var i=0;i<arr.length;i++){
        var tmp = levenshtein_distance(a,arr[i]);
        if(tmp<minD){
            result = arr[i];
            minD = tmp;
        }
    }
    return result;
}
```

Figure 38 calculate the minimum one

The result of this algorithm depends on developer's programming practice and naming practice.

4.2.3 Load user data from file

We load user data from file using Firefox directory and file service and then employ the Regular Expression to get the data `name and value from file`.

4.2.4 Script converter to java format

4.2.4.1 Adapter pattern

We adopt adapter pattern for formats to make the application extensible in future.

```
file = showFilePicker(window, "Select a File",

Components.interfaces.nsIFilePicker.modeOpen,Format.TEST_CASE_DIRECTORY_PREF
,function(fp) {return fp.file; });
```

Figure 39 Firefox file and directory service

```
var values =(lines[line]).match( /(?=\S)[^=]+?(?=\s*(=|$))/g );
```

Figure 40 regular expression

In the formatAdapterCommand class, we check the commands line by line. If the command has the category of 'fctn', it will call format page object function adapter and format function adapter. Otherwise, it will be formatted to either command or comment. And if we output the script as Java-web driver, the adapter will call the formatting functions in webdriver.js.

Following are part of the adapter functions:

```
function formatUIMapAdapter(testCase,name){
      var result = '';
      if (this.formatUIMap) {
        result = formatUIMap(testCase);
      }
      return result;
}
```

Figure 39 formatUIMapAdapter

```
function formatPageobjectAdapter(page,name) {
      this.log.info("formatting page object: " + name);
      var result = '';
      var header = "";
      var footer = "";
      this.commandCharIndex = 0;
      if (this.formatPageobjectHeader) {
        header = formatPageobjectHeader(page);
      }
      result += header;
      this.commandCharIndex = header.length;
      //testCase.formatLocal(this.name).header = header;
      result += formatFunctions(page.fctns); //Page should also have a
variable list
      if (this.formatFooter) {
        footer = formatPageobjectFooter(page);
      }
      result += footer;
      //testCase.formatLocal(this.name).footer = footer;
      return result;
}
```

Figure 40 formatPageobjectAdapter

4.2.4.4 Format to webdriver-junit4

After called adapter function, it will pass to the corresponding format type as you select. Followings are the

```
function formatPageobjectHeader(page){
    var className = page.name;
  className = testClassName(className);
  var header =options.pageobjectHeader.
      replace(/\$\{className\}/g, className).
    replace(/\$\{uiVariables\}/g,formatVars(page.targetPageVars)).
     replace(/\$\{uiVariablesWithValues\}/g,formatVariableWithValues
(page.targetPageVars,"UIMap.properties")).
      replace(/\$\{([a-zA-Z0-9_]+)\}/g, function(str, name) {
        return options[name];
      });
  this.lastIndent = indents(parseInt(options.initialIndents, 10));
  return header;
}
```

Figure 41 formatPageobjectHeader

```
function formatFunctionHeader(fctn){
  var result = "";
  result += "public void "+ fctn.name + "(";
  for ( var i=0;i < fctn.parameters.length-1;i++) {
    result  =  result + "String " + fctn.parameters[i].name + ",";
  }
  if (fctn.parameters.length) {
    result = result + "String " +
fctn.parameters[fctn.parameters.length-1].name;
  }
  result = result +"){";
  return result;
}
```

Figure 42 formatFunctionHeader

```
function callPageFctn(fctn,params){
  var page = fctn.page;
  var pageName = (page.name).toLowerCase();
  var paraSegment ='';
  for(var i=0;i<params.length-1;i++){
    paraSegment += params[i] + ',';
  }
  if (params.length>0) {
    paraSegment += params[params.length-1];
  }
  return  pageName + '.'+fctn.name +'(' + paraSegment + ')';

}
function newPageObject(page) {
  return page.name + ' '+ (page.name).toLowerCase() + ' = ' + 'new
' + page.name + '(driver)';
}
```

Figure 43 callPageFctn

4.2.5 Test case generator

Implementation of the algorithm to generate test case

```
var i=0;
 while(i<self.numK){
  worklist= this.getTestCase().valueVars;
  ti= worklist[i%worklist.length];
  testCase= clone(that);
  var v =[];
  D=0;
  for(var k=0;k<self.threshold;k++){
      eval( "v["+k+"] = "+"chance."+"string"+"()");
  }
  for(var j=0;j<self.threshold;j++){
    di =
levenshtein_distance(normalizeString(ti.value),normalizeString(v[j]))
    if (di>D) {
      D=di;
      t= v[j];
    }
  }
  testCase.updateVarRelationsART(true,ti.name,t);
  testCase.setTitle(thatTestcase.getTitle()+"dataSet"+i);
  this.app.getTestSuite().addTestCaseFromContent(testCase);
  this.app.setTestCase(testCase);
  this.playCurrentTestCase();
  if(this.getTestCase().result ==”passed”){
      worklist.push(t);
    }
  i++;
}
```

Figure 44 test case generator algorithm

Implementation to normalize the string:

```
function normalizeString(a){
      var result =[];
      for(var i =0;i<a.length;i++){
            if((/([A-Z])/g).test(a[i])){
              result.push(3);
            }else if((/([a-z])/g).test(a[i])){
               result.push(2);
            }else if ((/(\d)/g).test(a[i])) {
              result.push(1);
            }else if ((/[=<>+]/g).test(a[i])) {
              result.push(0);
            }

      }
      return result.join("");
    }
```

Figure 45 normalize string algorithm

Chapter 5 Result and Evaluation

5.1 Major components user interface and result:

5.1.1 Data variables

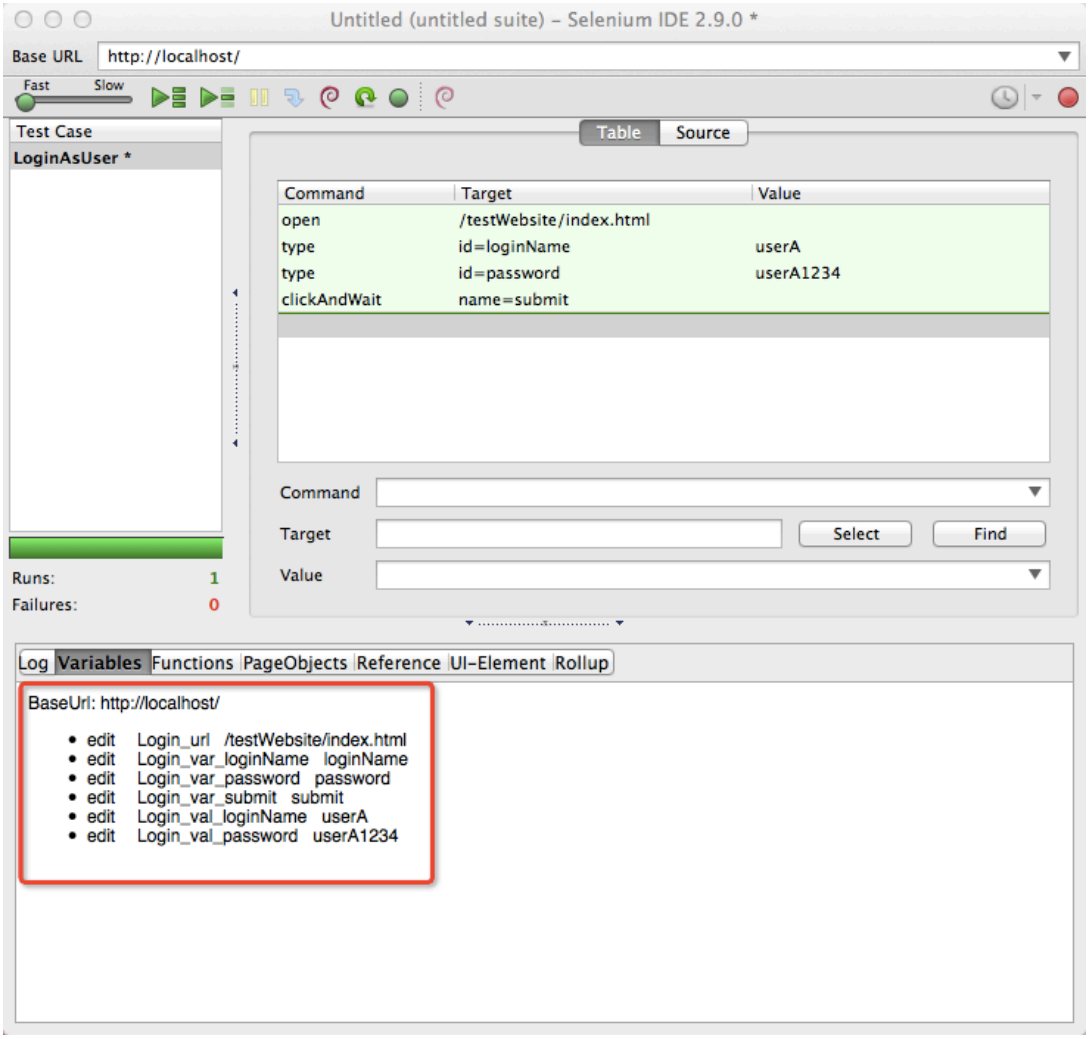


Figure 46 Data variable view

The UI elements and value elements converted into variables will be displayed in the variable views and tester can edit the variable name and edit it. The UI elements variable and value variables will be converted into a separate UI map file and value map files (Figure 47). And the value variables will be pushed into array and reused. Tester can reuse these variables when manually editing the test command and the value variables. For example, like in figure, tester can reuse login user name value to verify the user name displayed after login.

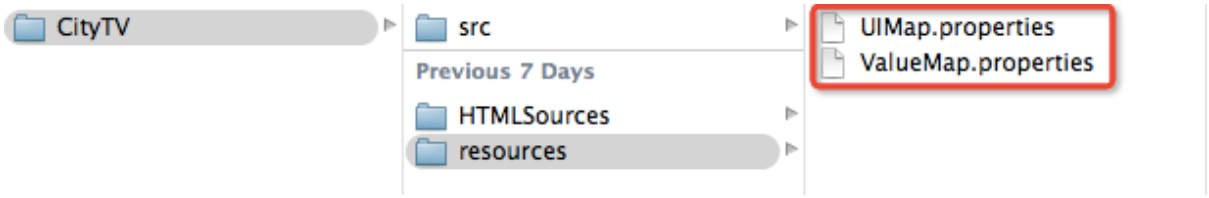


Figure 47 Property files

ValueMap.properties

```
Login_val_loginName = userA
Login_val_password = userA1234
CA1_Registration_page_val_userName = Mathilda Moore
CA1_Registration_page_val_contact = (431) 488-5331
CA1_Registration_page_val_address = 1014 Vicoj Manor
CA1_Registration_page_val_email = kewodo@furakavim.org
CA1_Registration_page_val_password = b!erTrc2Dh8XIl1
CA1_Registration_page_val_confirm = b!erTrc2Dh8XIl1
```

Figure 48 Value map file

UIMap.properties

```
Login_url = /testWebsite/
Login_var_loginName = loginName
Login_var_password = password
Login_var_submit = submit
Login_var_undefined = undefined
Login_url = /testWebsite/index.html
Login_var_Register_As_New_user = Register As New user
CA1_Registration_page_var_userName = userName
CA1_Registration_page_var_contact = contact
CA1_Registration_page_var_address = address
CA1_Registration_page_var_email = email
CA1_Registration_page_var_password = password
CA1_Registration_page_var_confirm = confirm
CA1_Registration_page_var_submit = submit
CA1_Profile_page_var_logout = logout
```

Figure 49 UI map file

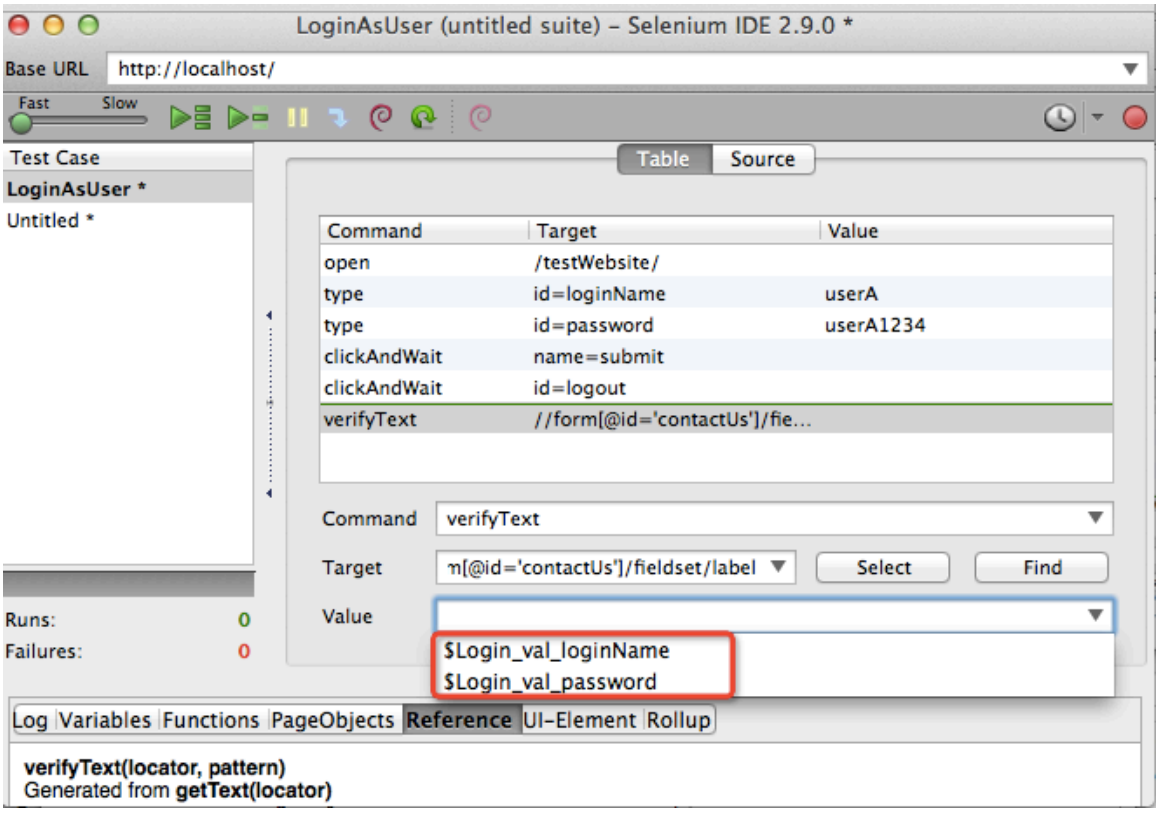


Figure 50 Auto completion of stored value variable when menu editing

5.1.2 Page object

Tester can add page object in PageObjects view. A page object does not necessary to be a whole page.

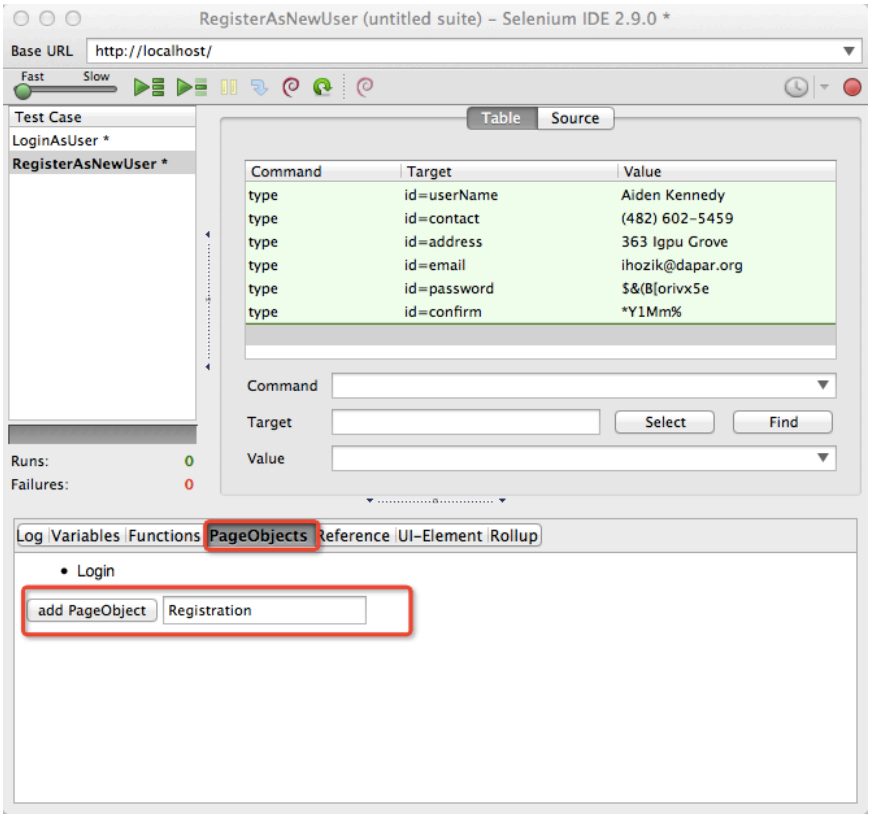


Figure 51 Page Objects view

5.1.3 Group command action:

It is inevitable that certain sequences of Selenium commands will appear in test cases over and over again. Moreover, tester can select multiple test steps and click the grouping button, then the steps is grouped into a function.

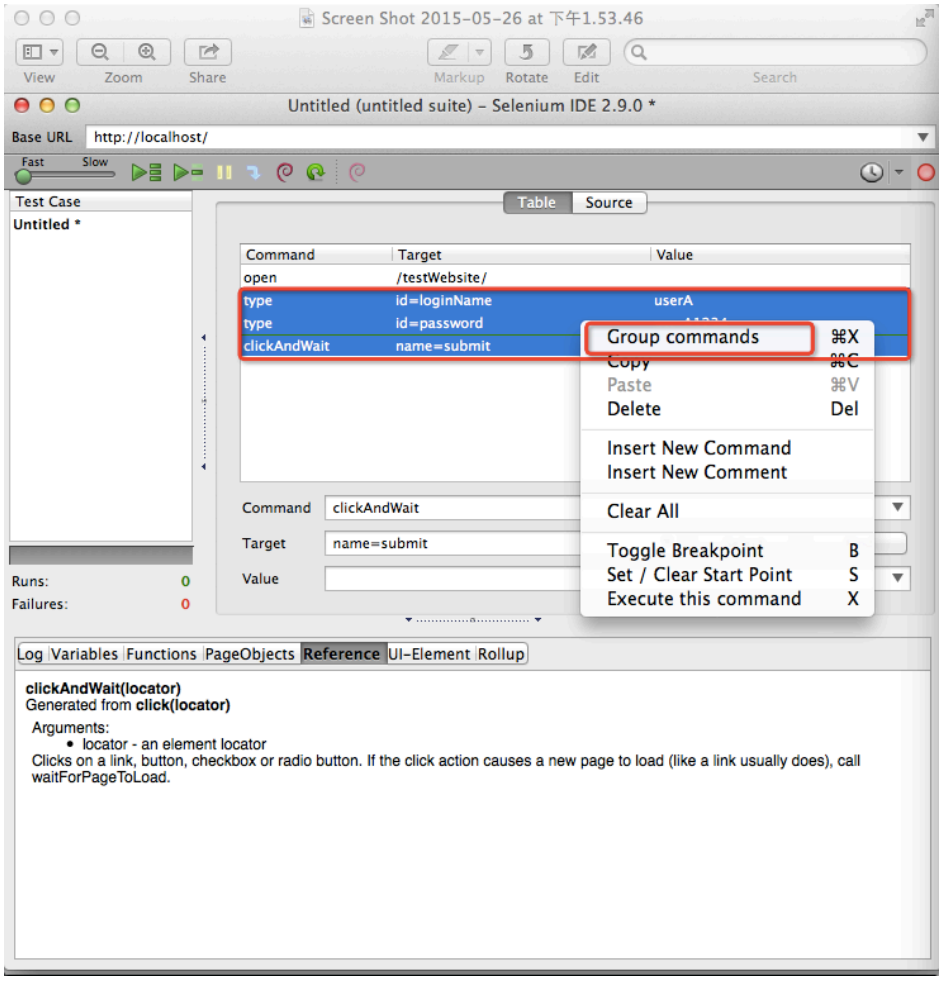


Figure 52 Group commands

After click 'Group commands', A dialog will pop up. Tester can either choose to group as existing function or create a new function (Figure 18).

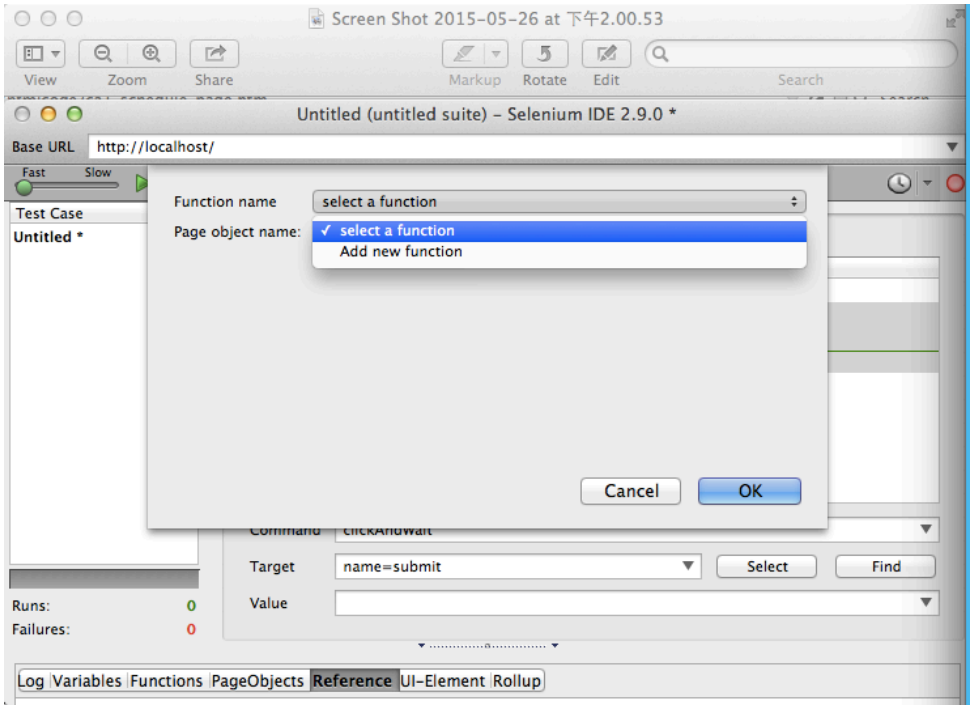


Figure 53 Select an existing function or create a new function

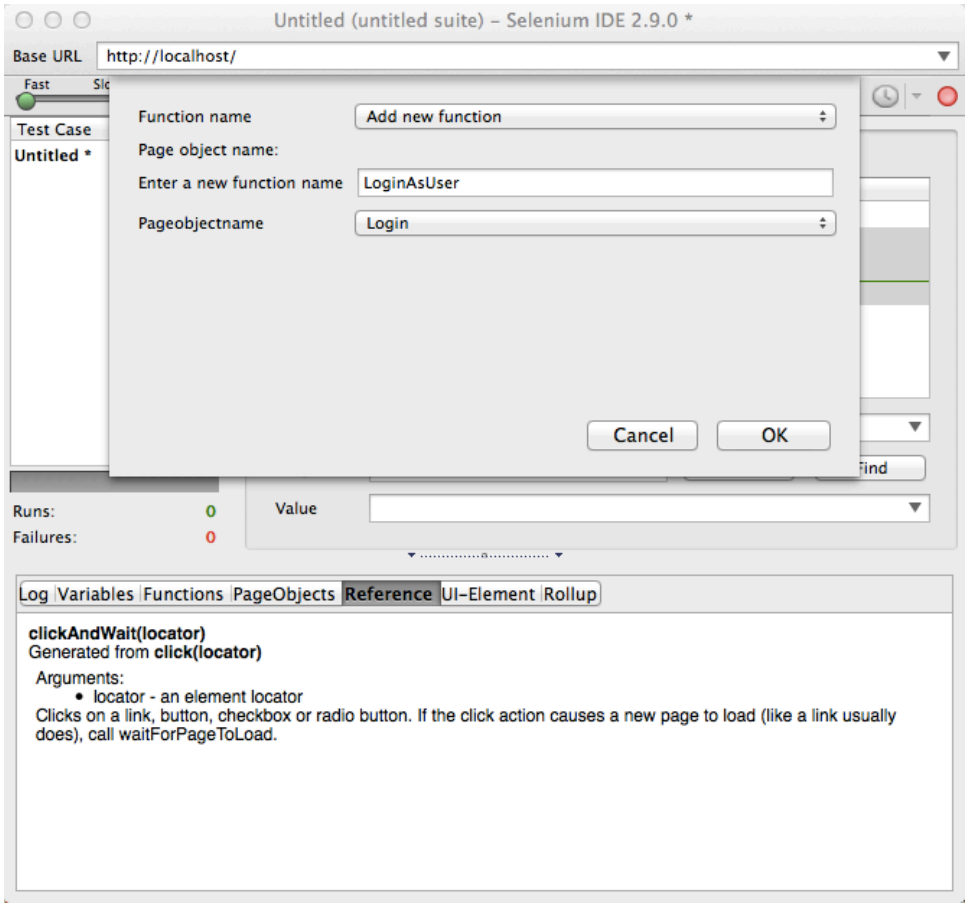


Figure 54 Select a page the function belong to

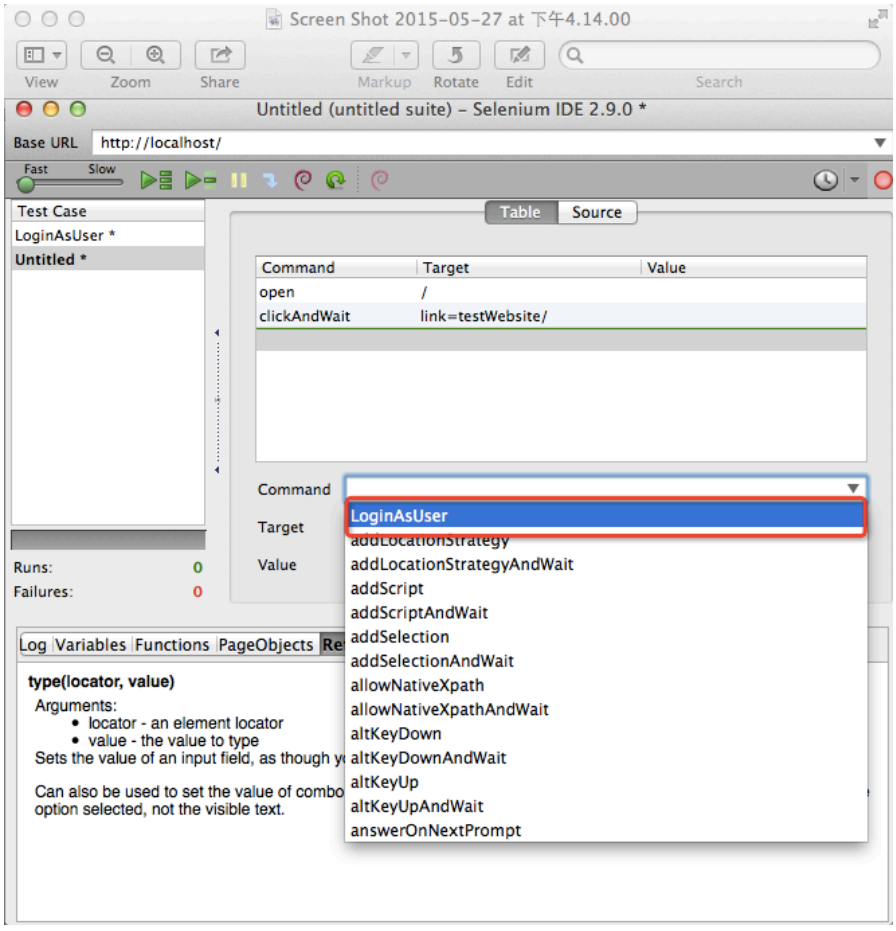


Figure 55 Stored function will be displayed in command auto-completion list

The new added functions are pushed to the command list. And you can reuse it across the test cases inside one test suite. After you select the function, it will be executed on the web application automatically (Figure 56).

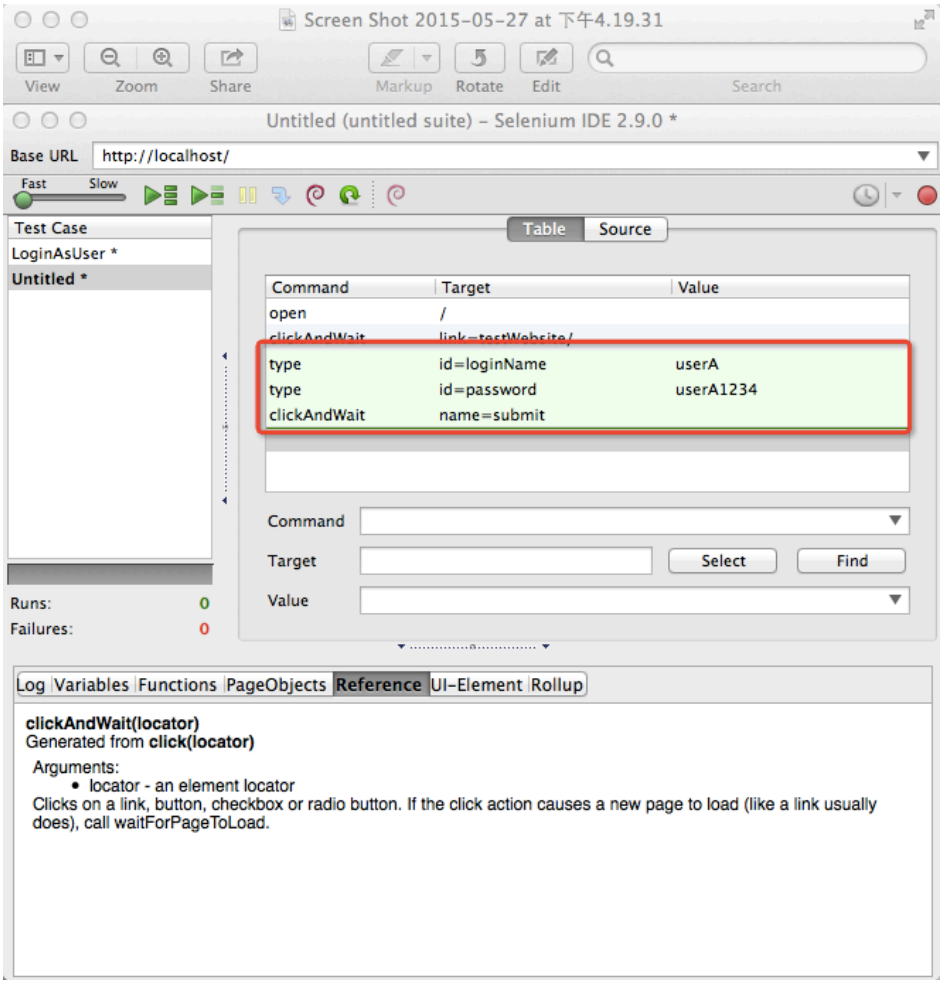


Figure 56 Function is expanded into commands in the tree view

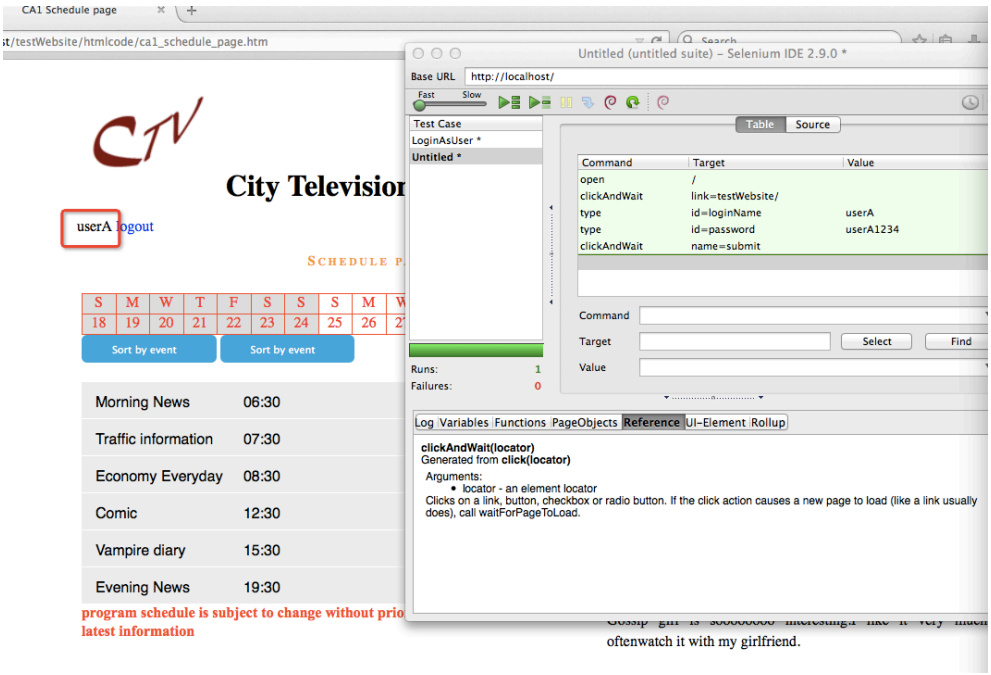


Figure 57 Login as userA without repeating the actions

And then tester can login as the user without repeating the actions.

Tester can also view the functions created in function view (Figure 58).

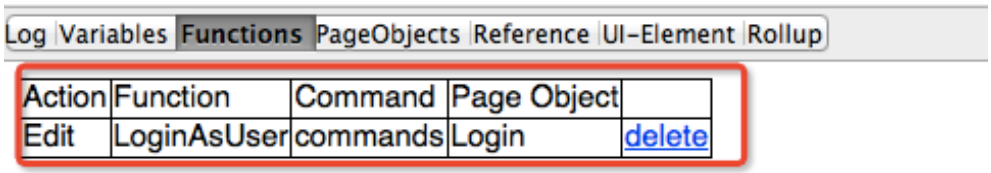


Figure 58 New created functions displayed in function view

5.1.4 Random test data generation

The system utilizes the predefined data pool provided by Chance.js. Part of the data pool from Chance.js (Figure 59,60,61)

```
    firstNames: {
        "male": ["James", "John", "Robert", "Michael", "William", "David", "Richard", "Joseph", "Charles", "Thomas",
"Christopher", "Daniel", "Matthew", "George", "Donald", "Anthony", "Paul", "Mark", "Edward", "Steven", "Kenneth", "Andrew",
 "Brian", "Joshua", "Kevin", "Ronald", "Timothy", "Jason", "Jeffrey", "Frank", "Gary", "Ryan", "Nicholas", "Eric",
"Stephen", "Jacob", "Larry", "Jonathan", "Scott", "Raymond", "Justin", "Brandon", "Gregory", "Samuel", "Benjamin",
"Patrick", "Jack", "Henry", "Walter", "Dennis", "Jerry", "Alexander", "Peter", "Tyler", "Douglas", "Harold", "Aaron",
"Jose", "Adam", "Arthur", "Zachary", "Carl", "Nathan", "Albert", "Kyle", "Lawrence", "Joe", "Willie", "Gerald", "Roger",
"Keith", "Jeremy", "Terry", "Harry", "Ralph", "Sean", "Jesse", "Roy", "Louis", "Billy", "Austin", "Bruce", "Eugene",
"Christian", "Bryan", "Wayne", "Russell", "Howard", "Fred", "Ethan", "Jordan", "Philip", "Alan", "Juan", "Randy", "Vincent"
, "Bobby", "Dylan", "Johnny", "Phillip", "Victor", "Clarence", "Ernest", "Martin", "Craig", "Stanley", "Shawn", "Travis",
"Bradley", "Leonard", "Earl", "Gabriel", "Jimmy", "Francis", "Todd", "Noah", "Danny", "Dale", "Cody", "Carlos", "Allen",
"Frederick", "Logan", "Curtis", "Alex", "Joel", "Luis", "Norman", "Marvin", "Glenn", "Tony", "Nathaniel", "Rodney",
"Melvin", "Alfred", "Steve", "Cameron", "Chad", "Edwin", "Caleb", "Evan", "Antonio", "Lee", "Herbert", "Jeffery", "Isaac",
"Derek", "Ricky", "Marcus", "Theodore", "Elijah", "Luke", "Jesus", "Eddie", "Troy", "Mike", "Dustin", "Ray", "Adrian",
"Bernard", "Leroy", "Angel", "Randall", "Wesley", "Ian", "Jared", "Mason", "Hunter", "Calvin", "Oscar", "Clifford", "Jay",
"Shane", "Ronnie", "Barry", "Lucas", "Corey", "Manuel", "Leo", "Tommy", "Warren", "Jackson", "Isaiah", "Connor", "Don",
"Dean", "Jon", "Julian", "Miguel", "Bill", "Lloyd", "Charlie", "Mitchell", "Leon", "Jerome", "Darrell", "Jeremiah", "Alvin"
, "Brett", "Seth", "Floyd", "Jim", "Blake", "Micheal", "Gordon", "Trevor", "Lewis", "Erik", "Edgar", "Vernon", "Devin",
"Gavin", "Jayden", "Chris", "Clyde", "Tom", "Derrick", "Mario", "Brent", "Marc", "Herman", "Chase", "Dominic", "Ricardo",
"Franklin", "Maurice", "Max", "Aiden", "Owen", "Lester", "Gilbert", "Elmer", "Gene", "Francisco", "Glen", "Cory", "Garrett"
, "Clayton", "Sam", "Jorge", "Chester", "Alejandro", "Jeff", "Harvey", "Milton", "Cole", "Ivan", "Andre", "Duane", "Landon"
],
```

Figure 59 first name pool

```
    months: [
        {name: 'January', short_name: 'Jan', numeric: '01', days: 31},
        // Not messing with leap years...
        {name: 'February', short_name: 'Feb', numeric: '02', days: 28},
        {name: 'March', short_name: 'Mar', numeric: '03', days: 31},
        {name: 'April', short_name: 'Apr', numeric: '04', days: 30},
        {name: 'May', short_name: 'May', numeric: '05', days: 31},
        {name: 'June', short_name: 'Jun', numeric: '06', days: 30},
        {name: 'July', short_name: 'Jul', numeric: '07', days: 31},
        {name: 'August', short_name: 'Aug', numeric: '08', days: 31},
        {name: 'September', short_name: 'Sep', numeric: '09', days: 30},
        {name: 'October', short_name: 'Oct', numeric: '10', days: 31},
        {name: 'November', short_name: 'Nov', numeric: '11', days: 30},
        {name: 'December', short_name: 'Dec', numeric: '12', days: 31}
    ],
```

Figure 60 months pool

```
// Data taken from https://github.com/umpirsky/country-list/blob/master/country/cldr/en_US/country.json
countries: [{"name":"Afghanistan","abbreviation":"AF"},{"name":"Albania","abbreviation":"AL"},{"name":"Algeria",
'abbreviation":"DZ"},{"name":"American Samoa","abbreviation":"AS"},{"name":"Andorra","abbreviation":"AD"},{"name":"Angola",
'abbreviation":"AO"},{"name":"Anguilla","abbreviation":"AI"},{"name":"Antarctica","abbreviation":"AQ"},{"name":"Antigua
and Barbuda","abbreviation":"AG"},{"name":"Argentina","abbreviation":"AR"},{"name":"Armenia","abbreviation":"AM"},{"name":
'Aruba","abbreviation":"AW"},{"name":"Australia","abbreviation":"AU"},{"name":"Austria","abbreviation":"AT"},{"name":
'Azerbaijan","abbreviation":"AZ"},{"name":"Bahamas","abbreviation":"BS"},{"name":"Bahrain","abbreviation":"BH"},{"name":
'Bangladesh","abbreviation":"BD"},{"name":"Barbados","abbreviation":"BB"},{"name":"Belarus","abbreviation":"BY"},{"name":
'Belgium","abbreviation":"BE"},{"name":"Belize","abbreviation":"BZ"},{"name":"Benin","abbreviation":"BJ"},{"name":"Bermuda"
,"abbreviation":"BM"},{"name":"Bhutan","abbreviation":"BT"},{"name":"Bolivia","abbreviation":"BO"},{"name":"Bosnia and
Herzegovina","abbreviation":"BA"},{"name":"Botswana","abbreviation":"BW"},{"name":"Bouvet Island","abbreviation":"BV"},{
'name":"Brazil","abbreviation":"BR"},{"name":"British Antarctic Territory","abbreviation":"BQ"},{"name":"British Indian
Ocean Territory","abbreviation":"IO"},{"name":"British Virgin Islands","abbreviation":"VG"},{"name":"Brunei","abbreviation"
:"BN"},{"name":"Bulgaria","abbreviation":"BG"},{"name":"Burkina Faso","abbreviation":"BF"},{"name":"Burundi","abbreviation"
:"BI"},{"name":"Cambodia","abbreviation":"KH"},{"name":"Cameroon","abbreviation":"CM"},{"name":"Canada","abbreviation":"CA"
},{"name":"Canton and Enderbury Islands","abbreviation":"CT"},{"name":"Cape Verde","abbreviation":"CV"},{"name":"Cayman
Islands","abbreviation":"KY"},{"name":"Central African Republic","abbreviation":"CF"},{"name":"Chad","abbreviation":"TD"},{
'name":"Chile","abbreviation":"CL"},{"name":"China","abbreviation":"CN"},{"name":"Christmas Island","abbreviation":"CX"},{
```


Figure 61 countries pool

The system crawler will extract the input box information and the system will analyze the value of input text box attribute such as name, id. Currently, if the system cannot produce all correct results or every input box and if the wrong data type, tester can correct it manually.

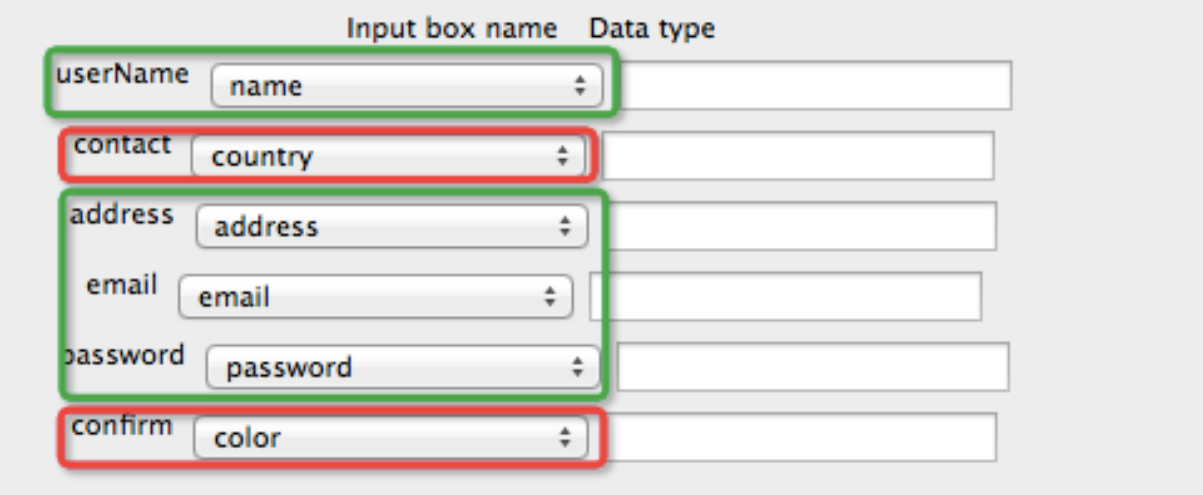


Figure 62 Data type matching result

After click ‘generate data’ button, you can preview the data in corresponding textbox (Figure 63).

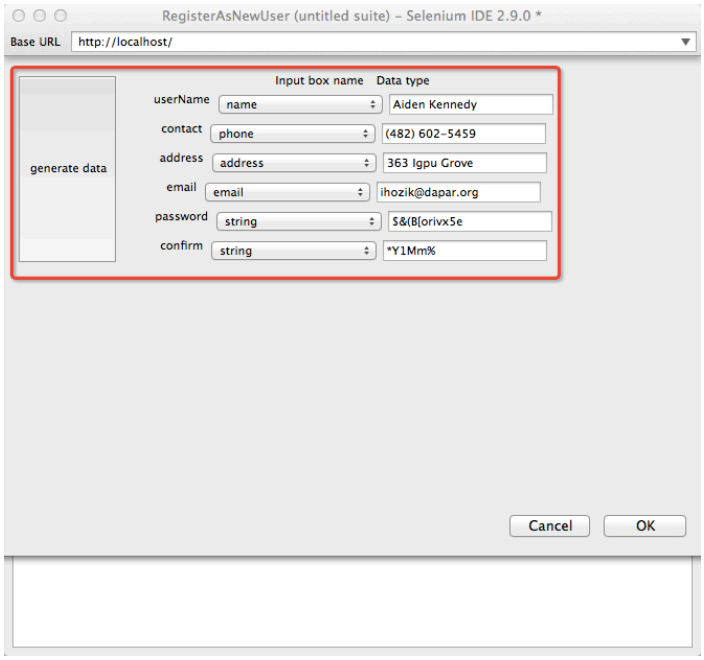


Figure 63 Generated data preview

And after you click 'OK', the system will add the commands into the tree view table (Figure 64) and fill the form with the generated data automatically (Figure 65).

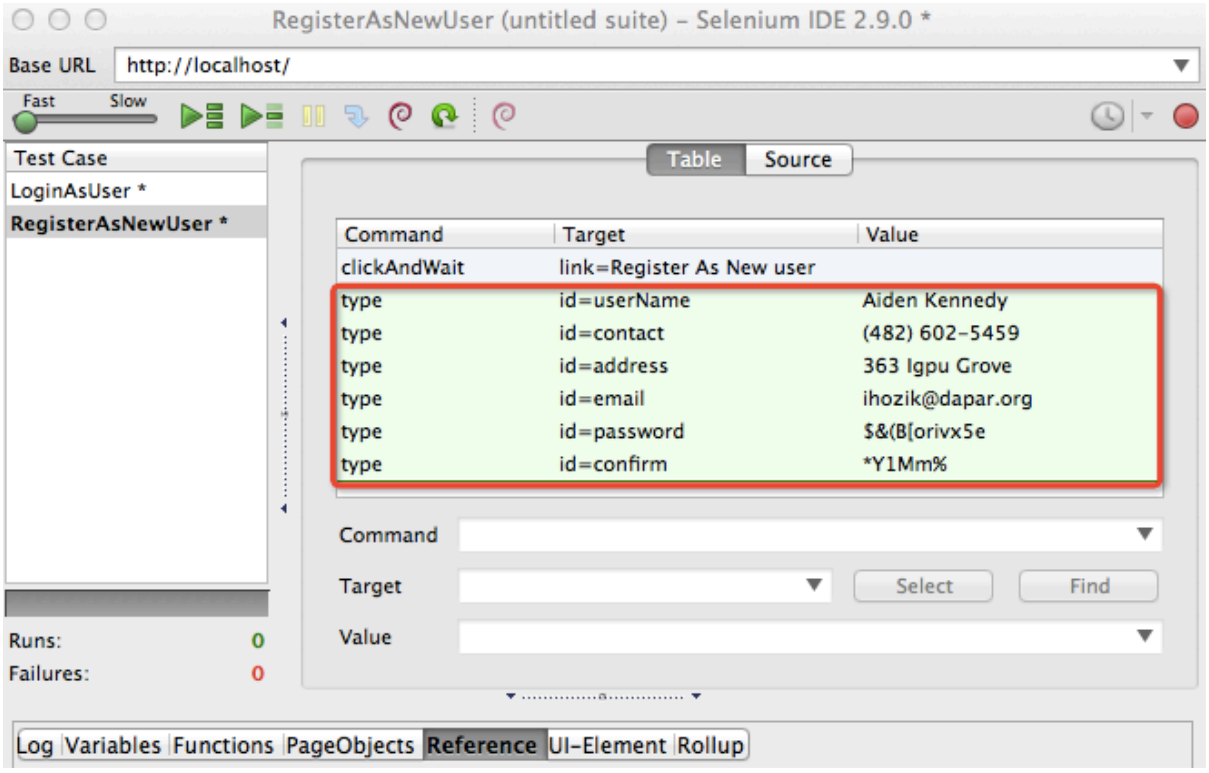


Figure 64 Generate commands into the tree view

Figure 65 Fill the form with the generated data automatically

These values are also stored for reuse, such that you can use it to verify the profile page.

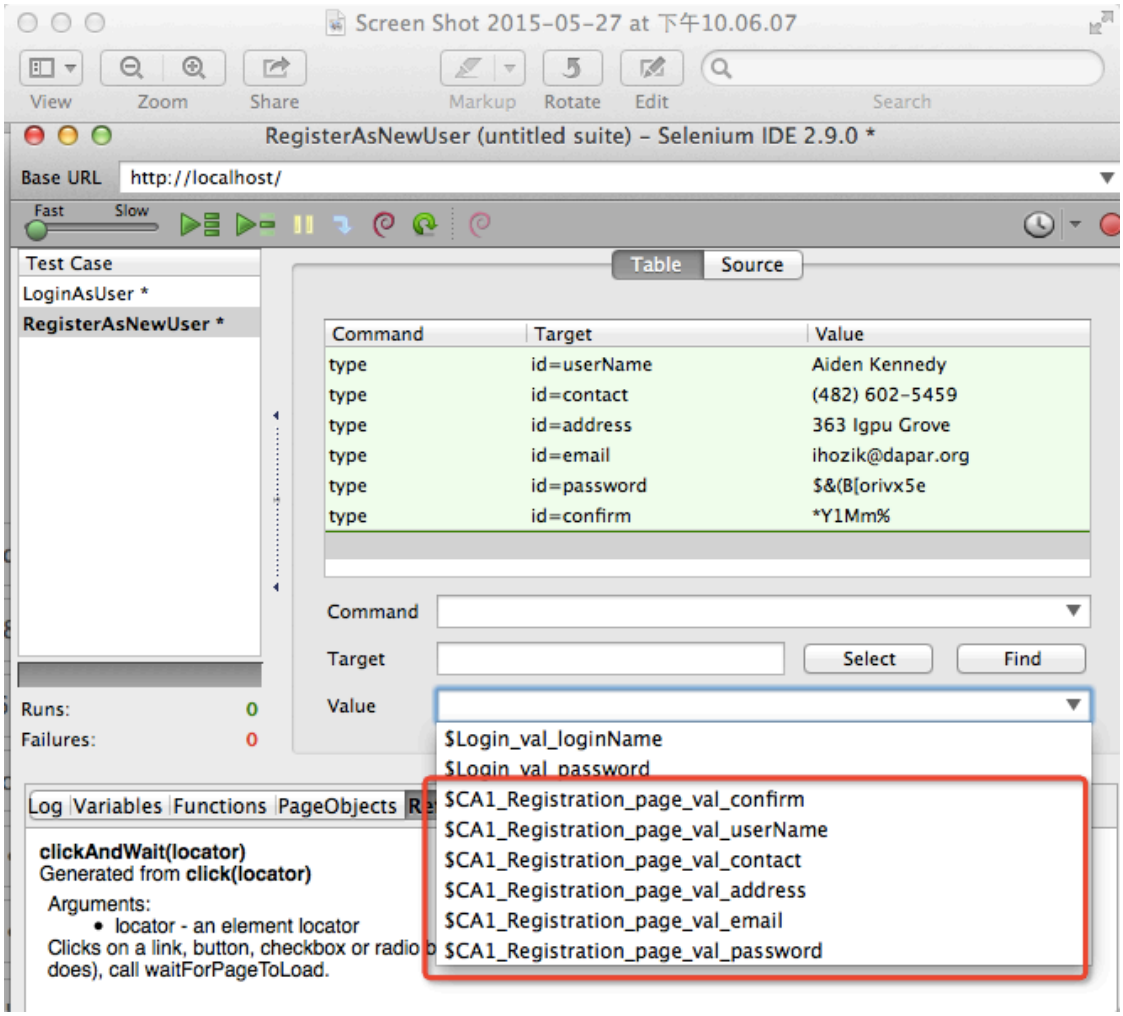


Figure 66 Value variable displayed in the auto completion list of the value

5.1.5 Get user test data

Load user value file:

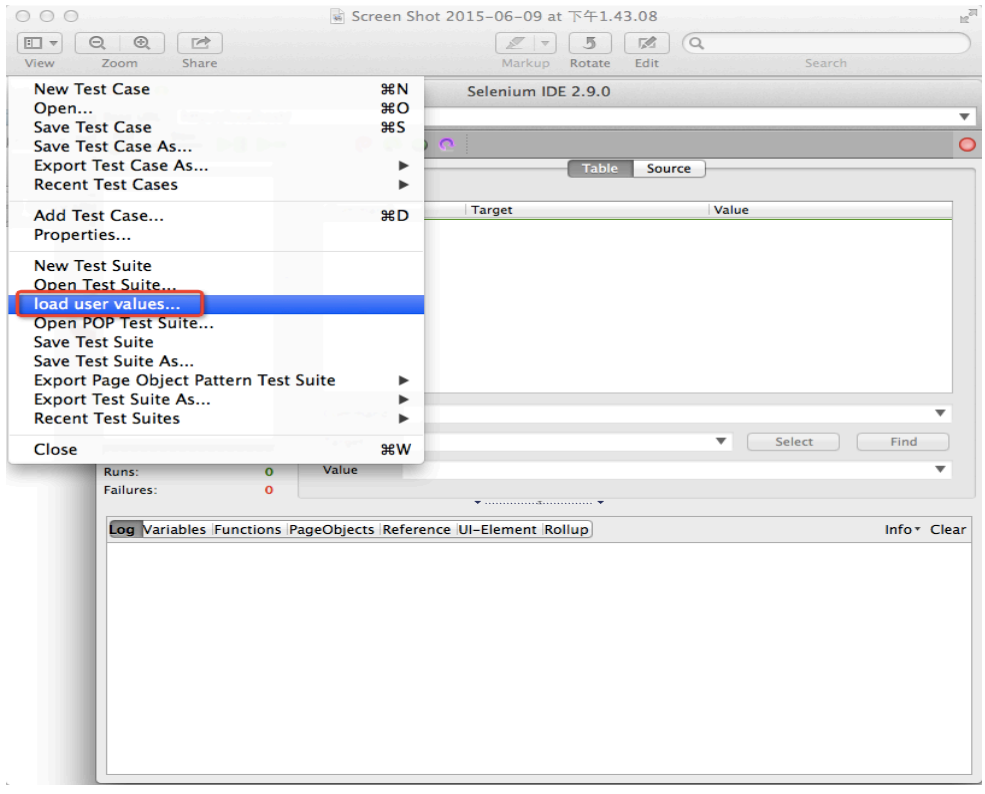


Figure 67 Load user value file

The contents of user value file:

ValueMap.properties

```
userName = Gene Fox
contact = (939) 374-5802
address = 1191 Fuino Drive
email = aw@rebzome.io
password = G^OGbbT&wHOU*HNH6U
confirm = G^OGbbT&wHOU*HNH6U
loginName = userA
loginPassword = userA1234
```

Figure 68 User value file

When testing form elements or input box, tester can get the user input value.

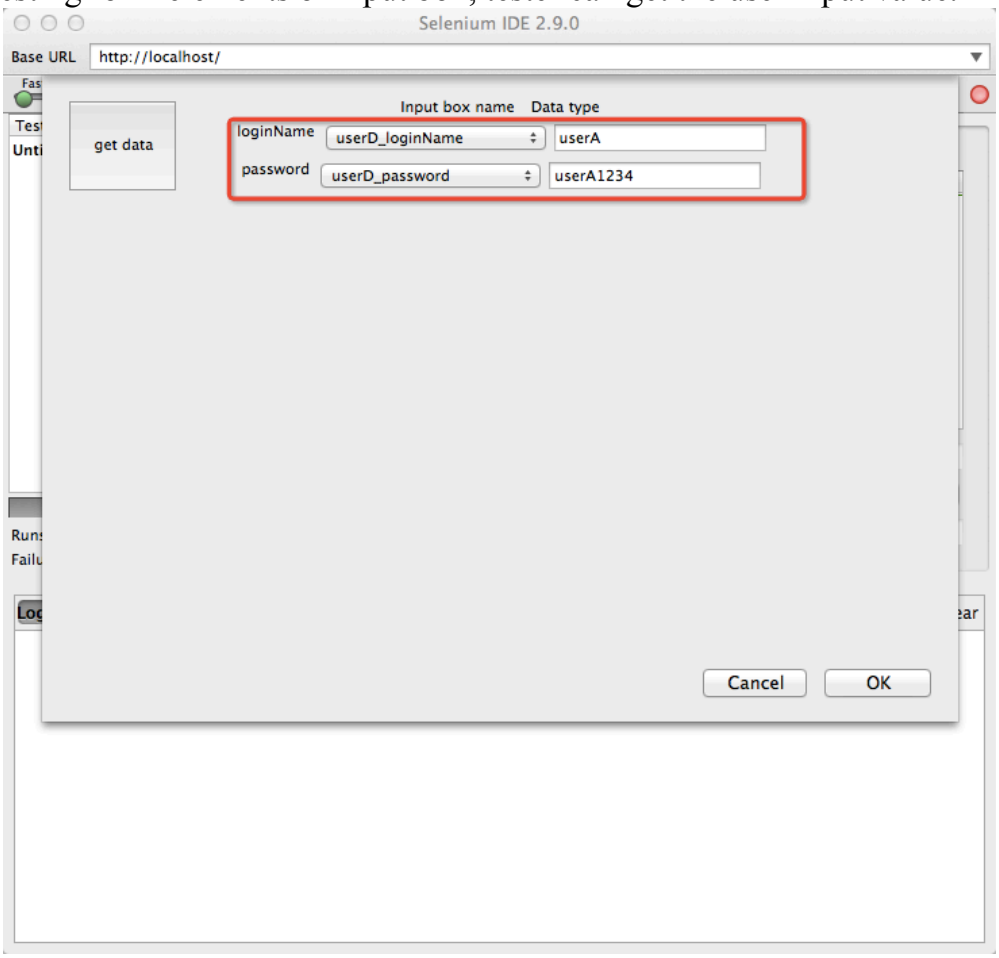


Figure 69 Get user data

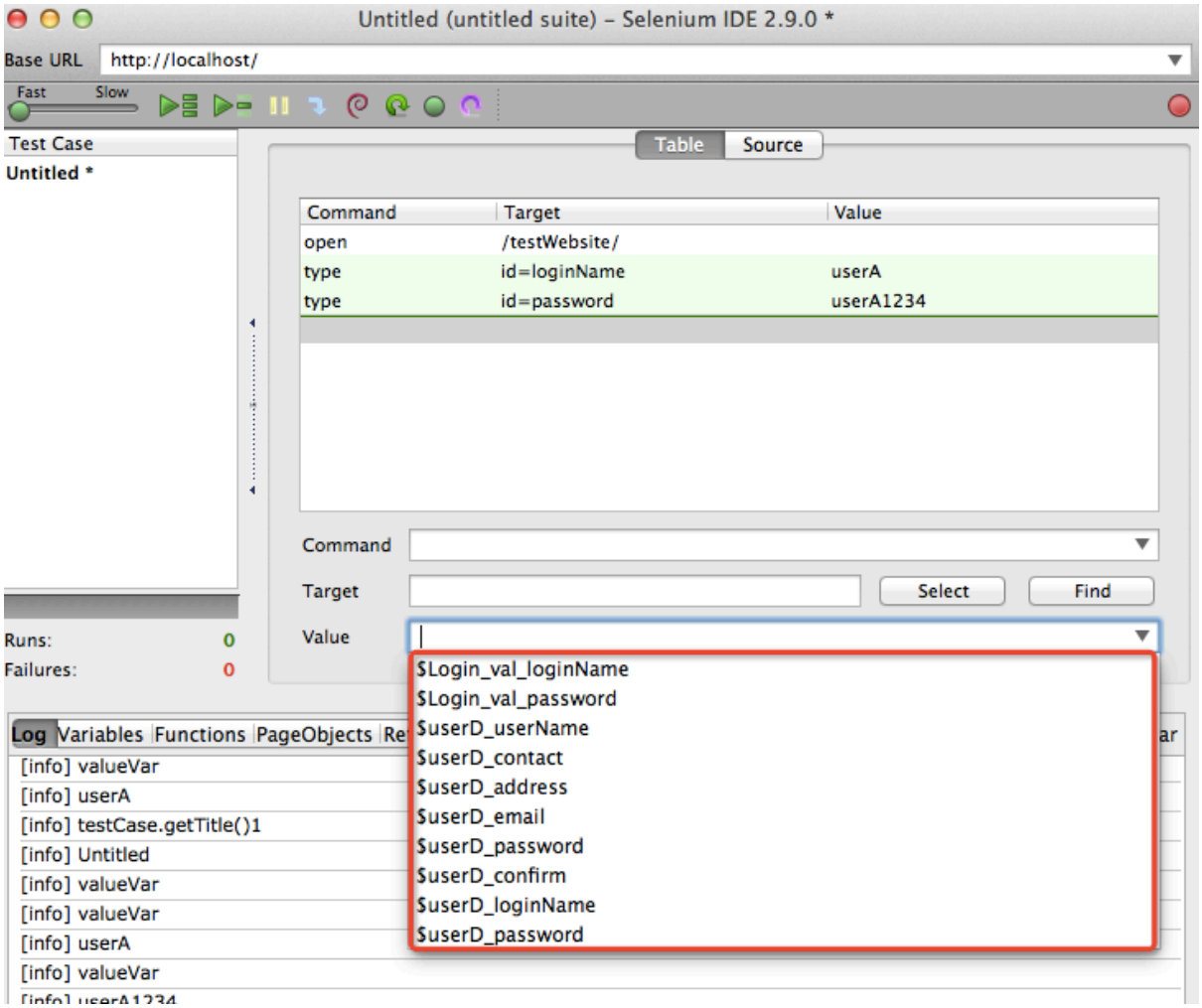


Figure 70 Get user data

5.1.6 Script converter

Script convertor is to refactor the test script (low level script output by selenium IDE) of an individual test case and put all test cases in structured and encapsulated test suite currently adopting page object pattern. The UI elements variables and user input data will be output to a resource file within the test suite (Figure 76). With the original Selenium IDE, test cases are independent to each other make the function not reusable (Figure 72). And user input data and UI elements are scaled across test cases (Figure 73). After processed by converter and applied Page Object Pattern, the code inside every single file will be reconstructed and the same logic like login is removed to the function under login page object (Figure 77). This could save considerable maintenance effort as the application scale up.

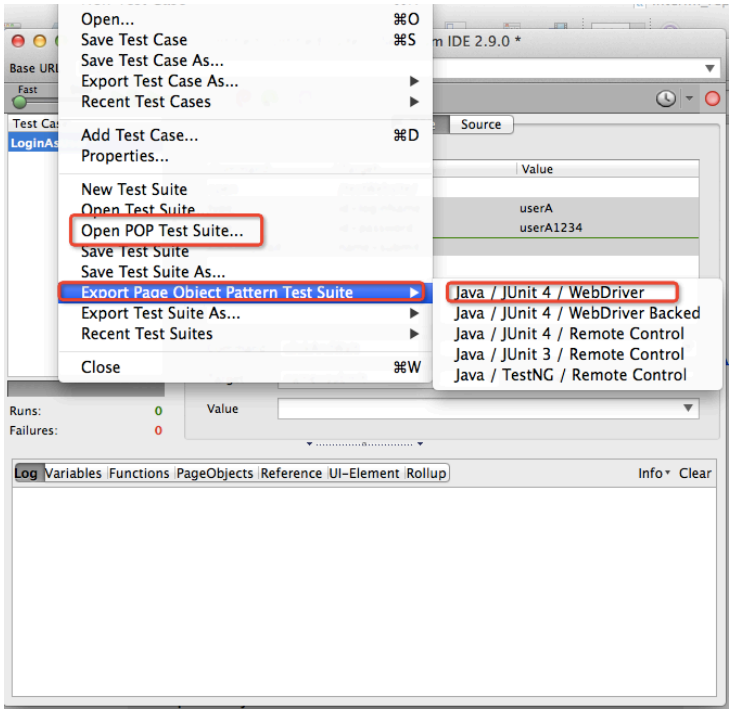


Figure 71 Output as Page object pattern test suite

Original selenium IDE output:

File structure:

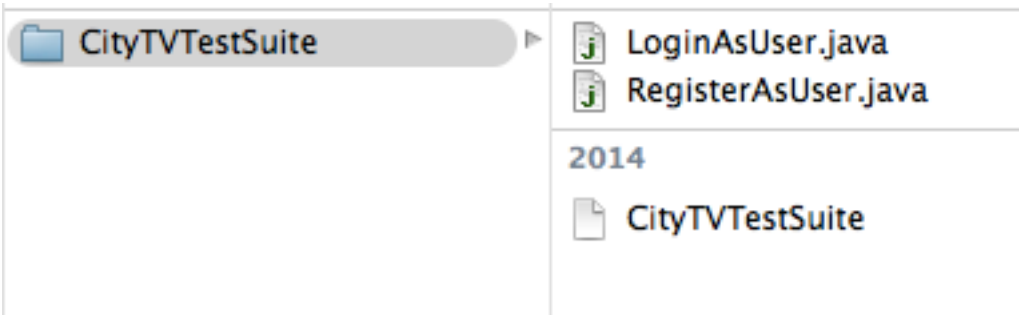


Figure 72 Test Suite file structure by original selenium ide

Individual test script of the test case output by selenium IDE

```
12 public class LoginAsUser {
13   private WebDriver driver;
14   private String baseUrl;
15   private boolean acceptNextAlert = true;
16   private StringBuffer verificationErrors = new StringBuffer();
17
18   @Before
19   public void setUp() throws Exception {
20     driver = new FirefoxDriver();
21     baseUrl = "http://localhost/fyptest/testWebsite/";
22     driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
23   }
24
25   @Test
26   public void LoginAsUser() throws Exception {
27     driver.get(baseUrl + "/fyptest/testWebsite/");
28     driver.findElement(By.name("fname")).clear();
29     driver.findElement(By.name("fname")).sendKeys("user");
30     driver.findElement(By.name("lname")).clear();
31     driver.findElement(By.name("lname")).sendKeys("user1234");
32     driver.findElement(By.cssSelector("input[type=\"button\"]")).click();
33
34   }
```

Figure 73 Individual test script

System output:

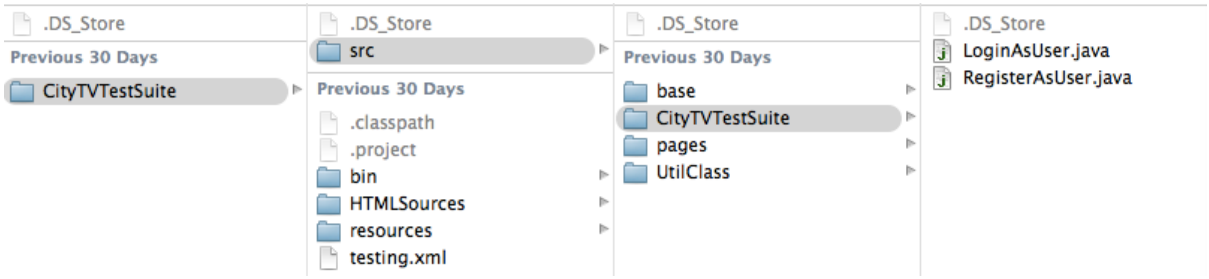


Figure 74 Test cases script

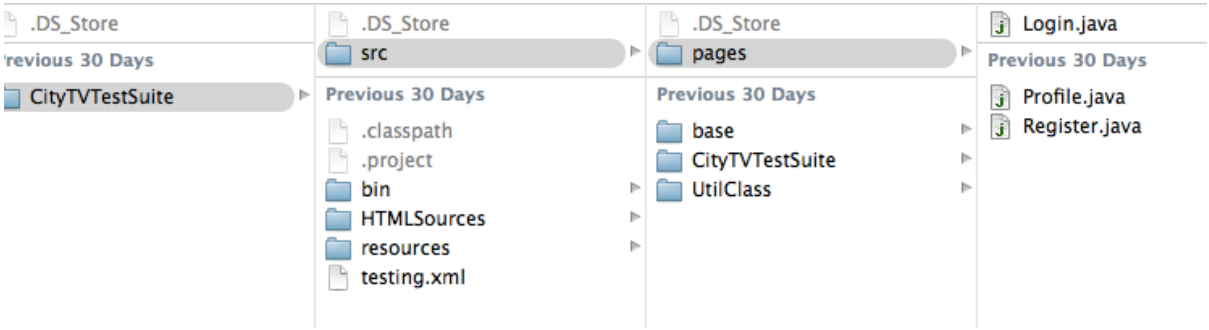


Figure 75 Page objects script

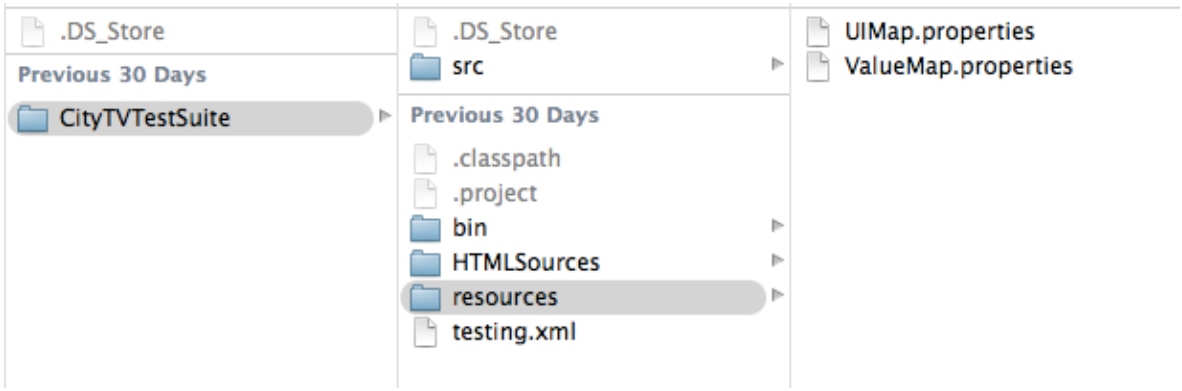


Figure 76 Test suite output by the project

Test script after the script converter

```
9  public class Login {
10   private WebDriver driver;
11   private String var_loginName;
12   private String var_password;
13   private String var_submit;
14
15   public Login(WebDriver driver) throws Exception {        Page object constructor
16     this.driver = driver;
17     UtilsFunctions uf = new UtilsFunctions();
18     Properties prop = uf.getPropertyFile("UIMap.properties");
19     var_loginName = prop.getProperty("var_loginName");
20     var_password = prop.getProperty("var_password");
21     var_submit = prop.getProperty("var_submit");
22                                                  Login function
23   }
24 public void LoginAsUser(String val_loginName,String val_password){
25     driver.findElement(By.id(var_loginName)).clear();
26     driver.findElement(By.id(var_loginName)).sendKeys(val_loginName);
27     driver.findElement(By.id(var_password)).clear();
28     driver.findElement(By.id(var_password)).sendKeys(val_password);
29     driver.findElement(By.name(var_submit)).click();
30 }
31 }
```

Figure 77 Test script after the script converter

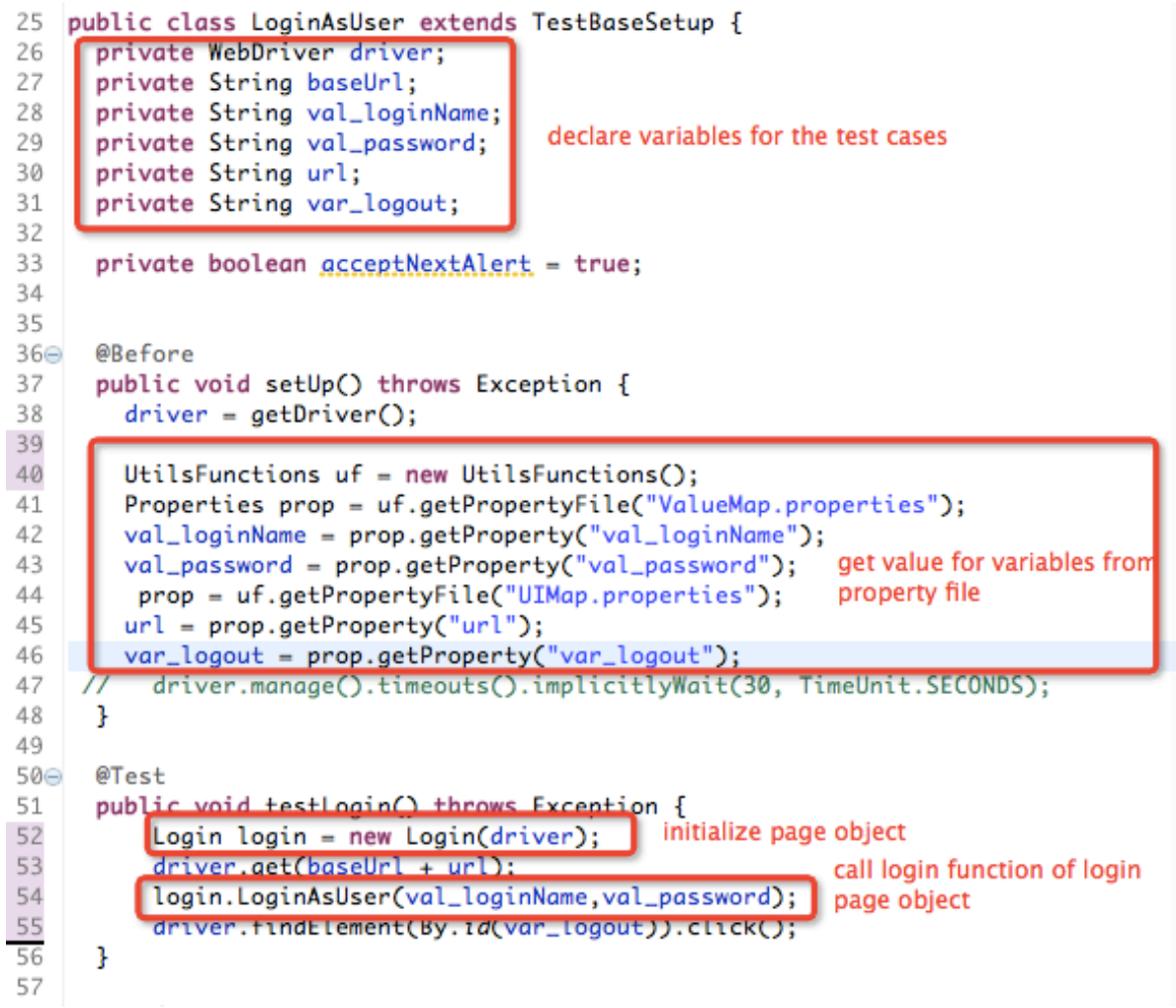


Figure 78 Test script after the script converter

5.1.7 Test case generator

5.1.7.1 Valid test data generator

In 3.3.4, the value variables will save the data type and the Positive test data generator will retrieve the data type and generate another set of positive random data, and test against the application once the data is generated. Normally the test case with positive data will pass. The failed test case on the left panel is because of the password is different from the confirm password which could also be use as negative test.

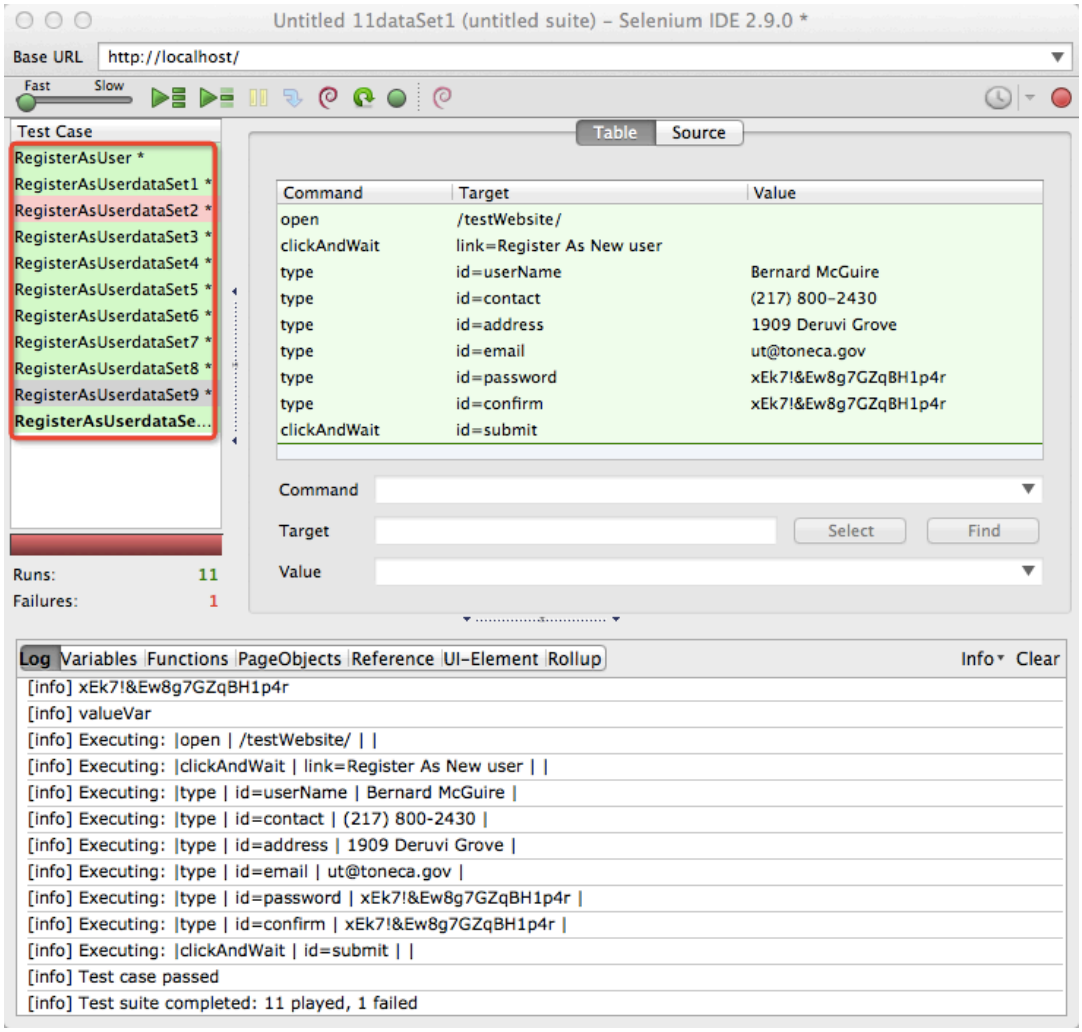


Figure 79 Generating test cases

5.1.7.2 Random test case generator

After tester click generate random test case, the system will generate the number of test cases and run it once generated. And on the left panel, the failed test case indicates failed result and green means passed. And we can double click the failed the test case to replay the error (Figure 82) and we can change the test case name to registerUserWithInvalidEmail Address. However, if tester checks the passed test cases, he could find a defect that the invalid contact data passed the test case.

The result could result from the following combination of results. According to the Table 14, at current stage, no matter the test case pass or fail, we need to check the result manually.

Input	AUT	Result
Valid	Correct	Pass
Invalid	Correct	Fail
Valid	Incorrect	Fail

Invalid	Incorrect	Pass/Fail

Table 14 Test case analysis table

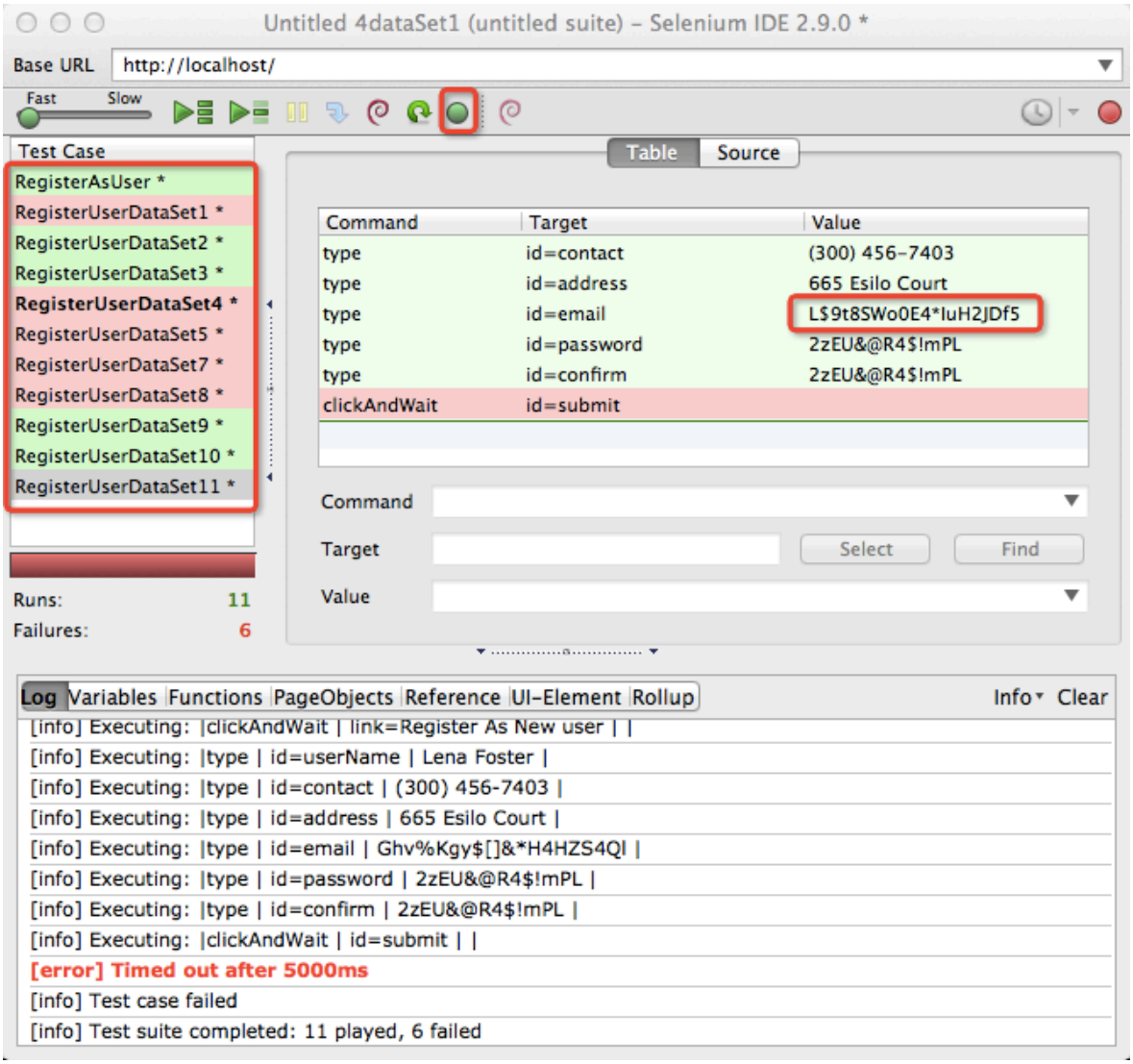


Figure 80 generate random test cases

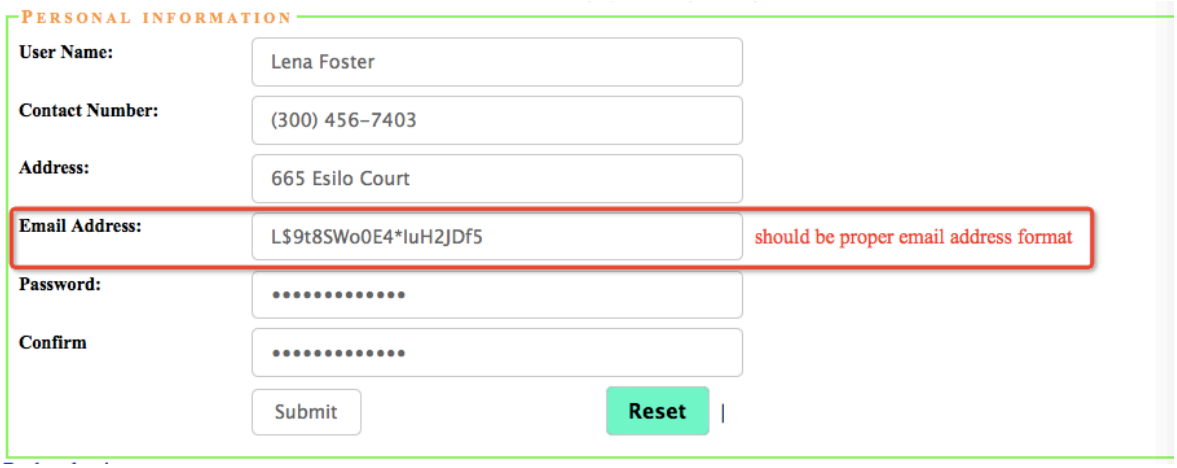


Figure 81 Invalid test data

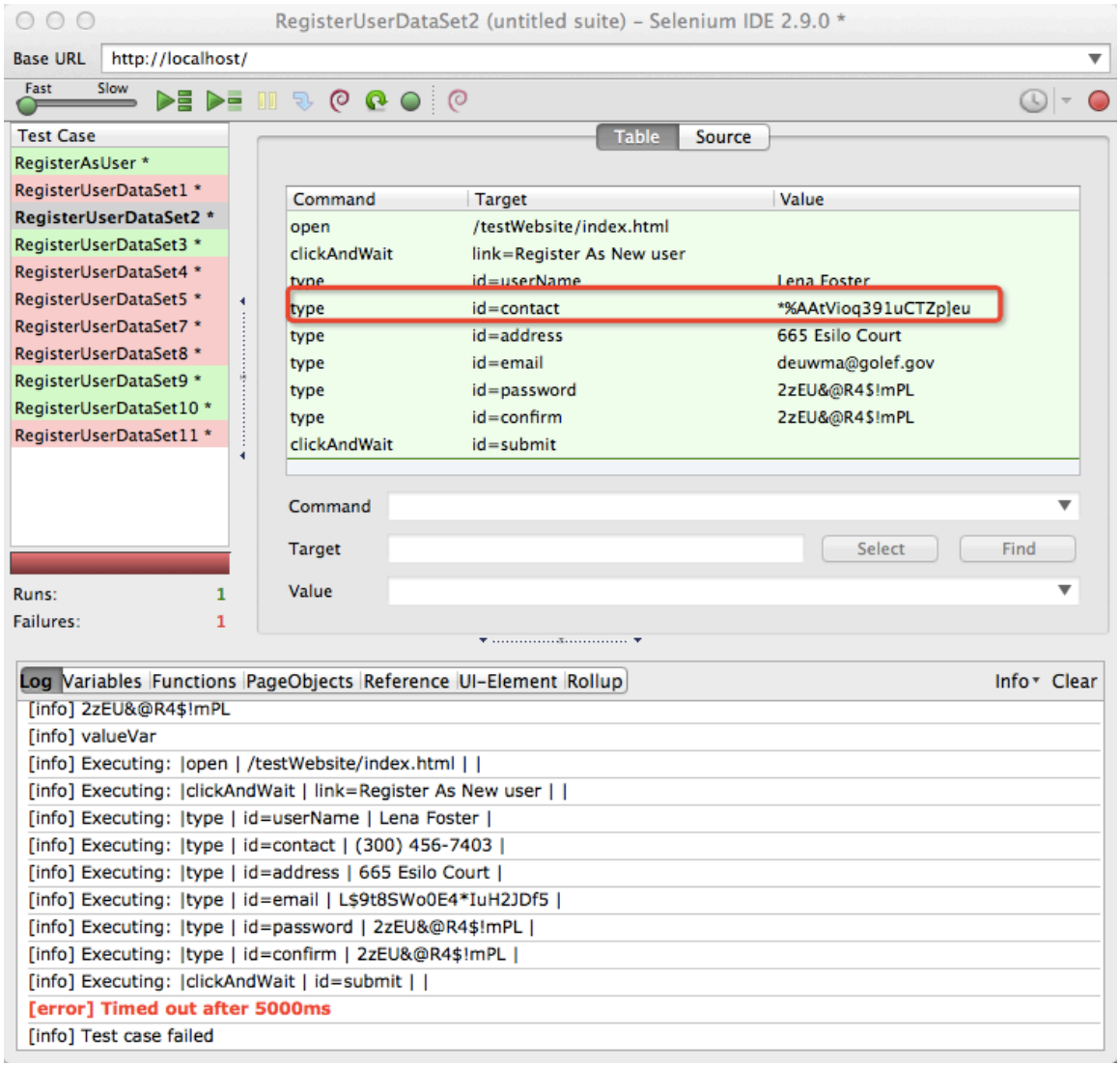


Figure 82 Invalid test data

5.2 Evaluation

We set three criteria to assess the system

1. Effort to construct an automated test suite instance
2. Following effort to maintain the resulting test suite
3. Ability to generate test data and test cases variety

We set up a scenario to test these three criteria. We would compare Original Selenium IDE, this system to construct an automated test suite. Automation tool like QTP is not open source and Watir doesn't have record and replay tool where you need to construct a test suite script from zero using Ruby script. There for we mainly compare the result with original selenium IDE.

Testing City TV

Summary:	
Target web application:	City TV
Web url:	http://localhost/testWebsite/

Test cases number	4
Browser:	Firefox, Chrome
Number of pages	4 Login page Schedule page Registration page Program page

Table 15

Test case 1 – register as user:		
	Test step	Expected result
1	Enter the website url	Log in page display
2	Click 'register as user' link	Switch to register user page
3	Fill in the name	Data display correctly
4	Fill in the phone	Data display correctly
5	Fill the address	Data display correctly
6	Fill the email address	Data display correctly
7	Fill the password	Data display correctly
8	Fill the confirm password	Data display correctly
9	Click submit	Switch to profile page
10	Verify the data display	User name, phone, address display correctly
11	Click log out	Switch to login page

Table 16

Test case 2 – Login as user:		
	Test step	Expected result
1	Enter the website url	Log in page display
2	Fill the user name	Data display correctly
3	Fill the password	Data display correctly
4	Click submit	Switch to schedule page
5	Verify the page title	Page tile: CA1 schedule page
6	Click profile page	Switch to profile page
7	Verify the data display	User name, phone, address display correctly
8	Click log out	Switch to login page

Table 17

Test case 3 – Login as admin:		
	Test step	Expected result
1	Enter the website url	Log in page display
2	Fill the admin name	Data display correctly
3	Fill the admin password	Data display correctly
4	Click submit	Switch to program page
5	Verify the page title	Page tile: CA1 program page

6	Click profile page	Switch to profile page
7	Verify the data display	User name, phone, address display correctly
8	Click log out	Switch to login page

Table 18

Test case 4 - Search a program		
	Test step	Expected result
1	Enter the website url	Log in page display
2	Fill the user name	Data display correctly
3	Fill the password	Data display correctly
4	Click submit	Switch to schedule page
5	Verify the page title	Page tile: CA1 schedule page
6	Click program page	Switch to program page
7	Type search text	Verify result display
8	Click log out	Switch to login page

Table 19

Criteria one assessment:

To construct a test suite contains 4 test cases:

Criteria /actions taken	Original Selenium IDE	Improved Selenium IDE
1	35	20

Table 20

Criteria two assessment:

Suppose we are going to change the base url, the steps you need to update the testing suite

Criteria /points to change	Original Selenium IDE	Improved Selenium IDE
2	4	1

Table 21

Criteria three assessment:

Improved Selenium IDE can generate negative test cases by data variation according to existing valid test cases

Criteria /test case generated	Original Selenium IDE	Improved Selenium IDE
3	0	10

Table 22

Chapter 6 Review and Future work

6.1 Review

6.1.1 Achievements:

In conclusion, the system provides various functions from different aspects to facilitate test automation process as I expected.

First of all, the system could extract the data from logic by extracting the value map and UI element map and make the test suite more configurable.

Secondly, the system could convert the output script into page object pattern which make the functional component reusable.

Moreover, the system can load user test data and previous generated data and generate directed random test data and random test cases, which indeed save considerable human effort.

6.1.2 Difficulties:

1. Time constraint
2. Firefox extension is hard to debug
3. Based on selenium IDE, need to study Selenium IDE code first

6.1.3 Limitations:

1. Testing have not been done on complicated web applications
2. Not fully automated and need some human interventions

6.2 Future work

6.2.1 Identity UI testing patterns of general web applications and generate test cases for general web applications

This is motivated by Moreira, R. M. et al. who propose six UI testing patterns namely Input UI test pattern, Login UI Test Pattern, Master/Detail UI test pattern, Find UI test pattern, Sort UI test pattern, Call UI test pattern. And each testing pattern has specific testing technique and testing logic. If we can define all web application components applicable to these generic UI testing patterns, then we only need to solve the testing

problem for these UI testing patterns which in turn solve the testing problem of most web applications.

6.2.2 Automatically generate candidate input by both modifying value and event

Currently, we only modify the value of test case to generate the test candidate input. As test case is made of a sequence of events, different combination of events also could result in unexpected defect and error. Therefore, in future, we would like to modify both event sequence and value of the test case ti, and then adopting the same algorithm as stated in section to find the most faraway test input ci from the ti. Since ci is modified from ti, ti should be the nearest neighbor of ci in the existing test suite.

6.2.3 Reduce failure test cases checking by identify the state change

As the most critical part is to reduce the duplicated and useless test input by random testing, we propose to identify the state of the failure point of a test input. We will mark the test input resulting in the same failure state as duplicated and being eliminated. As a result, user needs not to check so many test cases.

Chapter 7 Attachment:

7.1 Project Milestone

	Semester	Submission Date
Project plan	A	22 September 2014
Interim Report I	A	3 November 2014
Interim Report II	B	9 February 2015
Final Report	B	10 June 2015
Project Demonstration	B	12 June 2015

7.2 Project Schedule

ID	Task Name	Start	Finish
1	**Phase I-Project plan**	Mon 9/8/14	Mon 9/22/14
1.1	Define objectives, scope, deliverables Study existing system-Design and Implementation in Selenium IDE with Web		
1.11	Driver	Sun 9/21/14	Wed 9/24/14
1.2	Define milestones	Mon 9/8/14	Mon 9/8/14
1.3	Determine project risk Identify project risk and analyze	Mon 9/15/14	Mon 9/16/14
1.4	Define Schedule	Mon 9/8/14	Mon 9/22/14
1.5	Write project plan document	Mon 9/8/14	Mon 9/22/14
1.6	Monitor project progress	Right after completion of tasks	
2	**Phase II- Interim report I**		
2.1	Analyze existing system	Sun 9/21/14	Thur 9/24/14

2.1.1.2	Design system structure-prototyping and user scenarios	Tue 9/23/14	Thur 9/24/14
2.1.1.3	Review and revise design	Fri 9/25/14	Tue 9/30/14
2.1.1.5	Review progress and risk	Tue 9/30/14	Tue 9/30/14
2.1.2	Monthly Log	Tue 9/30/14	Tue 9/30/14
2.1.3	Interim report I 1. Introduction	Mon 9/29/14	Mon11/3/2014
2.1.3.1	2. Literature review	Mon 9/29/14	Mon 10/20/14
2.1.3.2	Review and revise	Tue 10/21/14	Mon 11/3/2014
2.1.3.3	Deliverables- Interim report I	Mon11/3/2014	Mon 11/3/2014
2.1.3	**Phase III - Interim report II**	Mon 11/10/14	Mon 2/9/2015
2.2.4.1	Research existing paper of testing techniques and testware structure	Mon 12/29/14	Fri 1/2/15
2.2.4.2	Design classes and algorism	Mon 1/5/15	Thur 1/15/15
2.2.4.3	Review and revise	Thur 1/15/15	Thur 1/23/15
2.2.1.4	Deliverables- class design and algorism design	Thur 1/23/15	Thur 1/23/15
2.2.1.5	Review progress and risk	Thur 1/23/15	Thur 1/23/15
	First draft-A final title & abstract for the project Introduction Literature review Proposed design, solution, system Detailed methodology and implementation Preliminary result and future improvement		
2.1.3.1	Monthly logs must be attached	Mon 11/10/14	Mon 1/12/15
2.1.3.2	Review and revise	Mon 1/12/15	Mon 2/9/15
	First draft-A final title & abstract for the project		
2.1.3.1	Introduction	Mon 11/10/14	Mon 2/9/15

	Literature review		
	Proposed design, solution, system		
	Detailed methodology and implementation		
	Preliminary result and future improvement		
	Monthly logs must be attached		
2.1.3.3	Deliverables- Interim report II	Mon 2/9/15	Mon 2/9/15

Phase III Final report

	Task6- Detailed implementation on		
2.2.5	crawler, function grouper with sqlDB	Tue 2/10/15	Fri 2/20/15
2.2.5.2	Code development	Tue 2/10/15	Thur 2/19/15
	Unite testing and integrate testing	Tue 2/10/15	Fri 2/20/15
	Deliverables-elements and functions can be		
2.2.5.3	stored in databases	Tue 2/10/15	Fri 2/20/15
2.2.5.4	Review progress and risk	Tue 2/10/15	Fri 2/20/15
2.3	Task 7 – Code converter	Tue 2/20/15	Tue 2/27/15
	Code development	Tue 2/20/15	Tue 2/27/15
	Unite testing and integrate testing	Tue 2/20/15	Tue 2/27/15
	Deliverables-Converter to covert original script to functional structured test scripts		
	Task7- Review testing techniques and detail		
2.3.6	implementation on test case generator	Sat 2/28/15	Wed 3/15/15
	Researching on the algorism of different		
2.3.6.1	test case generator techniques	Sat 2/28/15	Mon 3/2/15
2.3.6.2	Design classes and algorism	Mon 3/2/15	Tue 3/3/15
	Code development	Tue 3/3/15	Sun 3/15/15
2.3.6.3	Unite testing and integrate testing	Fri 3/13/15	Sun 3/15/15
2.3.6.3	Deliverables-test case generator	Sun 3/15/15	Sun 3/15/15
2.3.6.4	Review progress and risk	Sun 3/15/15	Sun 3/15/15
	Task7- Convert the new generated test		
2.3.6	cases to test script	Sun 3/15/15	Sat 3/21/15
2.3.6.2	Design classes and algorism	Sun 3/15/15	Mon 3/16/15

	Code development	Mon 3/16	Sat 3/21/15
2.3.6.3	Unite testing and integrate testing	Sun 3/15/15	Sat 3/21/15
2.3.6.3	Deliverables-converter	Sun 3/15/15	Sat 3/21/15
2.3.6.4	Review progress and risk	Sun 3/15/15	Sat 3/21/15
	System softcopy	Sun 3/22/15	Sun 3/22/15
	Final report		
	1. STRUCTURE & CONTENTS		
	1.1 General		
	1.2 Report Components		
	1.3 Industry-related Projects		
	2. DOCUMENT PREPARATION		
	CONVENTIONS		
	2.1 General		
	2.2 Abbreviations		
	2.3 Tables and Diagrams		
2.3.3	2.4 References	Mon 2/9/15	4/13/2015
2.3.3.1	First report draft	Mon 3/22/15	Mon 3/29/15
			Wed
2.3.3.2	Second report draft & Demo	Wed 3/29/15	6/10/2015
	Deliverables-		
	3. SUBMISSION PROCEDURE		
	3.1 Report Hardcopy		
	3.2 Softcopies		
	3.2.1 Report Softcopy		
	3.2.2 System Softcopy		Thur
2.3.3.3	3.2.3 Demo Softcopy	Thur 6/11/2015	6/11/2015
3	Phase III - Project Presentation	Fri 6/12/2015	

6.3 RISK MANAGEMENT

Hazard	Likelihood	Impact	Risk Exposure
Changes to requirements specification during coding	5	8	40
Sickness of project conductor	2	7	42
Module coding takes longer than expected	4	7	28
Module development too difficult to complete	3	10	30
Communication difficulties if project conductor is not in HK in Semester B	9	3	27

Since the project is to be done by individual and the project is much skills and knowledge dependent, so the risk exposure could be very high. Risk exposure is the product of likelihood and impact. Therefore, to decrease the risk exposure, we should ether prevent the risk item from happening or reduce the impact the item could result in.

Risk items Identification and Evaluation

Likelihood: 0-10, and 10 is 100% happen

Impact: 0-10, and 10 is highest impact

Planning Risk Management Activities and Implementation

ID	Software risk management action	Schedule
R1	Prototype project applications and prepare scenarios for complicated modules	System analysis stage
R2	Discuss with processionals like supervisor about changing the project scope	Right after the risk happen
R3	Allow buffer for the project process	Project scheduling stage
R4	Discuss with professionals like supervisor about changing the requirement	Right after the risk happen
R5	Plan regular online meeting	Before Angela leaving Hong Kong

6.4 Monthly log

September	Review literature and existing approach
October	A Demo web site under test was constructed, however the functionality should be more complete to show what the project could solve. And 3 demo test cases have been written to demonstrate problems, solutions,

	and expected output.
November	Finished interim report I and reviewed literature and existing approach on crawling techniques. User scenerio and use case is almost finalized and studyed crawling technique and test model and test patterns.
December	Reviewed literature and started to implement basic user interface and studied the technology to be used to constructed the application such as XUL, overlay.js.
January	Reviewed literature on test generating techniques and had more progress on the implementation and write interim report II.
February	Study Selenium IDE system
March	Complete the following functions: 1. Value Variables maps 2. UI Variables maps 3. Page Objects 4. Functions 5. Group command as function 6. Folder structure 7. Convert the test script in page Object pattern in Java using selenium web driver 8. Add/Edit value variables and UI variables functions and page objects
April	Complete the following functions 1. Crawl the input text box elements 2. The application can detect the input box elements and then generate random data from database or by function for different data type schema like person , contact, email, name, address(the first time) to facilitate user develop test 3. Now the system can run the test case once new test cases are generated and check the result 4. Autocomplete to load the stored value variables
May	Generate different random data adopting the merit of adaptive random testing technique and feedback random testing

	Final report
June	Final report, demo clip and presentation

References:

Amalfitano, D., Fasolino, A. R., & Tramontana, P. (2011). A GUI crawling-based technique for android mobile application testing. *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, (pp. 252-261).

Ammann, P.E., Knight, J.C. (1988). Data diversity: an approach to software fault tolerance. IEEE Transactions on Computers 37 (4), 418–425.

Artzi, S., Dolby, J., Jensen, S. H., Moller, A., & Tip, F. (2011). A framework for automated testing of JavaScript web applications. *Software Engineering (ICSE), 2011 33rd International Conference on*, (pp. 571-580).

Bauersfeld, S. (2013). GUIdiff--A regression testing tool for graphical user interfaces. *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, (pp. 499-500).

Bishop, P.G., 1993. The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail). In: Digest of Papers, the 23rd International Symposium on Fault- Tolerant Computing (FTCS-23). IEEE Computer Society Press, Los Alamitos, CA, pp. 98–107.

Cai, K.-Y., Zhao, L., Hu, H., & Jiang, C.-H. (2005). On the test case definition for GUI testing. *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*, (pp. 19-26).

Chen, T. Y., Kuo, F. C., Merkel, R. G., & Tse, T. H. (2010). Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, *83*(1), 60-66.

Cunha, M., Paiva, A. C., Ferreira, H. S., & Abreu, R. (2010). PETTool: a pattern-based GUI testing tool. *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, *1*, pp. V1--202.

Dogan, S., Betin-Can, A., & Garousi, V. (2014). Web application testing: A systematic literature review. *Journal of Systems and Software , 91*, 174-201.

Ernst, M. D., Perkins, J. H., Guo, P. J., McCamant, S., Pacheco, C., Tschantz, M. S., & Xiao, C. (2007). The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, *69*(1), 35-45.

Garousi, V., Mesbah, A., Betin-Can, A., & Mirshokraie, S. (2013). A systematic mapping study of web application testing. *Information and Software Technology , 55* (8), 1374-1396.

Huang, S., Cohen, M. B., & Memon, A. M. (2010). Repairing GUI test suites using a genetic algorithm. *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, (pp. 245-254).

Leotta, M., Clerissi, D., Ricca, F., & Spadaro, C. (2013). Improving test suites maintainability with the page object pattern: an industrial case study. *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, (pp. 108-113).

Li, Y.-F., Das, P. K., & Dowe, D. L. (2014). Two decades of Web application testing-A survey of recent advances. *Information Systems , 43*, 20-54.

Liu, Z., Gao, X., & Long, X. (2010, April). Adaptive random testing of mobile application. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on* (Vol. 2, pp. V2-297). IEEE.

Manning, C. D., Raghavan, P., & Sch{\"u}tze, H. (2008). *Introduction to information retrieval* (Vol. 1). Cambridge university press Cambridge.

Memon, A. M.(2001). *A comprehensive framework for testing graphical user interfaces.* Ph.D. dissertation, University of Pittsburgh.

Memon, A. M. (2002). GUI testing: Pitfalls and process. *Computer , 35* (8), 87-88.

Memon, A. M., & Xie, Q. (2005). Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. *Software Engineering, IEEE Transactions on , 31* (10), 884-896.

Memon, A. M., & Soffa, M. L. (2003). Regression testing of GUIs. *ACM SIGSOFT Software Engineering Notes*, *28*, pp. 118-127.

Memon, A., Banerjee, I., & Nagarajan, A. (2003). GUI ripping: Reverse engineering of graphical user interfaces for testing. *2013 20th Working Conference on Reverse Engineering (WCRE)*, (pp. 260-260).

Mesbah, A., & Van Deursen, A. (2009). Invariant-based automatic testing of AJAX user interfaces. *Proceedings of the 31st International Conference on Software Engineering*, (pp. 210-220).

Mesbah, A., Bozdag, E., & van Deursen, A. (2008). Crawling Ajax by inferring user interface state changes. *Web Engineering, 2008. ICWE'08. Eighth International Conference on*, (pp. 122-134).

Moreira, R. M., Paiva, A. C., & Memon, A. (2013). A pattern-based approach for GUI modeling and testing. *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, (pp. 288-297).

Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, *33*(1), 31-88.

Nedyalkova, S., & Bernardino, J. (2013). Open source capture and replay tools comparison. *Proceedings of the International C* Conference on Computer Science and Software Engineering*, (pp. 117-119).

Nguyen, B. N., Robbins, B., Banerjee, I., & Memon, A. (2014). GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering , 21* (1), 65-105.

Pacheco, C., & Ernst, M. D. (2005). *Eclat: Automatic generation and classification of test inputs.* Springer.

Pacheco, C., Lahiri, S. K., Ernst, M. D., & Ball, T. (2007). Feedback-directed random test generation. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, (pp. 75-84).

Paiva, A. C., Faria, J. C., Tillmann, N., & Vidal, R. A. (2005). A model-to-implementation mapping tool for automated model-based GUI testing. In *Formal Methods and Software Engineering* (pp. 450-464). Springer.

Saxena, P., Akhawe, D., Hanna, S., Mao, F., McCamant, S., & Song, D. (2010). A symbolic execution framework for javascript. *Security and Privacy (SP), 2010 IEEE Symposium on*, (pp. 513-528).

Uppal, N., & Chopra, V. (2012). Design and Implementation in Selenium IDE with Web Driver. *International Journal of Computer Applications , 46* (12).