National College of Ireland

Cloud Testing:Enhancing the speed of Test Automation using Cucumber Framework

MSc Research Project
Cloud Computing

Srujana Kaleru
x15040402

School of Computing
National College of Ireland

Supervisor: Dr. Adriana Chis

National College of Ireland
Project Submission Sheet – 2015/2016
School of Computing

Student Name:	Srujana Kaleru
Student ID:	x15040402
Programme:	Cloud Computing
Year:	2016
Module:	MSc Research Project
Lecturer:	Dr. Adriana Chis
Submission Due Date:	16/08/2017
Project Title:	Cloud Testing:Enhancing the speed of Test Automation using Cucumber Framework
Word Count:	5680

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	15th September 2017

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Cloud Testing:Enhancing the speed of Test Automation using Cucumber Framework

Srujana Kaleru
x15040402
MSc Research Project in Cloud Computing

16th August 2017

Abstract

In the current trend of Cloud Computing world, most of the companies are showing interest on developing open source projects that ultimately targets wide variety of browsers device users ranging from IE on Windows, Safari on Mac, Microsoft Edge on Windows 10, Firefox and chrome on all supported devices and operating systems. Before releasing the product into market, the company has to focus on testing of the product as it plays an important phase in Software Development Life Cycle to ensure the software product is free from defects by covering various types of testings that include functional, regression and cross browser compatibility . To initiate this testing phase and make the software as cloud enabled application huge investment is required in terms of hardware infrastructure to compensate the testing coverage on various browser platforms. To overcome this financial costs for companies, a cloud-hosted testing platform tools like sauce labs and BrowserStack is introduced in the market to speed up the automated testing of developed application. From the sources on the internet it is disclosed that currently 800 browser instances of beta versions with different O/S combinations are instantly available on the cloud-hosted testing platforms that eventually eases the hassle of infrastructure management for testing the applications. To use this infrastructure a reusable test automation framework is necessary for the testers to the run the tests automatically on different browsers. In the existing frameworks like keyword-driven and hybrid framework, tester need to have coding knowledge in order to adapt them. So the main intention of this research is to propose a re-usable automation framework where a non-technical tester can run the tests on the cloud hosted testing platform just by enabling the changes in feature files of cucumber. Furthermore, it is important to evaluate the amount of execution time taken by the tests on different browsers along with the memory consumption during each test run as this statistics will help us in identifying the proposed test automation framework will speedup the execution tests.

1 Introduction

Software testing is a process of validating a software with the goal to compare the deviations among provided input and intended output. In general, testing process is considered to be an important phase in the SDLC(Software Development Life Cycle) which

enables us to find the any errors or software defects with regard to the actual result. Software testing also facilitates for a quantitative analysis in finding the quality of any software feature in the software product. According to the IEEE 1059 standard, Testing is defined and characterized as "A process for evaluating the software to determine the variations between the existing and expected conditions (like any failures, defects and imperfections) and eventually to observe the software characteristics". Most of the companies will employ a separate team who are responsible for performing software testing as per the requirement specifications. Although the developer can perform unit testing for a developed feature during the development phase but overall testing coverage cannot be fulfilled during SDLC. As the number of business requirements increases, there will be extra effort on development department as only limited resources are responsible to build a product that aims for quality in short span of time. According to Wei-Tek et al[26] STA (software Test Automation) is defined as the process of automating the testing activities by using testing tools to develop the test scripts and execute them to perform functional verification based on the requirement specifications. As per Arif et al [2] explained that total cost incurred for manual testing is around 70% and for completing the software project takes around 50% time. They have also mentioned that testing a software is highly challenging as it includes vast coding of programming language, hardware and software requirements. Therefore to fill this gap an automatic software testing tool will be very helpful to minimize the cost and timing efforts for software development. However, there are few available automated testing tools in the market but offered them as licensed tools which start-up company could not afford them. So these start-up companies relies on open source automation tools such as Selenium, Robot Framework, Sahi, iMacros, QF-Test but these tools requires extensive programming knowledge to automate Graphical User Interface test cases. To test the database of web based application Castro et al.1 have suggested a new function by incorporating to the core library of selenium's Framework that helps to make a connection to the database for performing open and close operation. It is also successful in fetching the details of tested data to compare it with the stored data utilized by the application.However this method has minimized the testing effort process because the verification of UI and database will be invoked at same time when the test script execution process is started but unfortunately this framework did not functioned properly if the application database holds similar data.

Nowadays most of companies are developing the applications to provide the data access on wide variety of devices and desktop versions. For this process cross browser testing plays a crucial role as to check whether the application is behaving uniformly on all web browsers without any loss of content. Especially with the case of agile development that runs the sprint just for 4-5 weeks, testers are required to run the regression suites manually against different browsers needs very huge efforts. In order to test the same functional test scenarios rigorously for each sprint a well framed automation framework is highly needed to reduce the manual efforts. However there are many frameworks but none of them are very easy to understand and control by the non-programmers. Furthermore to test the application behavior on a large variety of web browsers on various operating systems needs lot of infrastructure and to maintain further human efforts are required.

In this research work, we have proposed re-usable test automation model that addresses the aforementioned problems in particular a framework that is highly understandable by the non-technical testers for easy maintenance and also that supports the cloud testing. Below are the major contributions of our proposed work,

1. A feature specific reusable framework that automates different applications.

2. With the aim to run the tests in cloud-hosted testing platform we integrated our re-usable automation framework with the Sauce Labs for performing cross-browser testing.

3. To fetch the results after each run we integrated a test reporting feature for our framework.

4. Collected the memory consumption and browser execution time after each test run are then recorded in tabular format as to analyze which browser is suitable for running the regression tests in the agile methodology .

In the remaining paper we have presented various automation frameworks under Related works and focused on the limitations of n-tiered automation testing framework, GRAFT framework approach. In the methodology we have discussed the architecture diagrams of Sauce lab integration with the framework followed by the internal components of framework on how the test-scripts are segregated based on Given, When and Then keywords. In the implementation we have presented on how the demo model of our reusable automation framework is build by integrating the sauce labs and test reports feature.In the evaluation section, we have shown that our re-usable framework can be used for automating multiple applications.In addition we collected the execution times taken by three different browsers after each test run and fetched the memory consumption that helps in identifying the memory leaks . Eventually we have proposed the future scope of the work in the automation testing area.

2 Related Work

We have studied various existing automation frameworks presented in various research articles. To understand the steps involved in automating any test-scenario for the web application is classified into four different types a) programming specific automation b) record the test-scenario and replay it c) image snapshot specific automation and d) domain specific automation. The major challenge with the existing frameworks that are proposed by Persson et al[15] and Micallef et al[12] involves code development work and is quiite difficult for the software testers . Leotta et al[6] introduced Image capturing based technology that will perform the Record and Replay actions which requires huge manual effort than programming specific approach . Little et al[7] proposed another image recognition tool like sikuli have shown a noticeable requirement for capturing the modification in browser resolution and also background effects . Thummalapenta et al[25] proposed ATA in which the scripts are presented in domain specific language and parsed into combinations of actions and web objects. R.Stejskal et al[23] demonstrated on open source tool namely concordion for devloping and maintaining automation based Acceptance tests in Java programming language. For performing regression and functional testing on various software based applications T.Lalwani et al[24]explained QTP (Quick Test Professional) is best license-based tool. It also has User Inteface through which tester can perform the testing. C.Rankin et al[16] proposed STAF that provides re-usable services. R.Mugridge et al[13] suggested FIT is the most suitable tool for performing acceptance testing as it contains general english instructions that establish communication among different stakeholders in agile based environment. A.Holmes et al[5] demonstrated that Selenium test scripts can be developed in table format and then later coded into any desired programming language such as ruby, Java, C and python which represents code

export capability that will eventually invoke the respective test scripts on the specified browsers. M.Yalla et al[27] has suggested that the combination approach of a re-usable automation framework and any open source tool is the best way and effective approach to minimize the testing time, efforts and time. Here most of the focus is on development of script less framework through Selenium RC. Also the framework facilitates easy methods for remote execution from the test scripts that were generated.

2.1 N-tiered Test automation Framework Approach :-

Most of the software development companies will segregate the design process of software project into different tiers typically termed as n-tiered design approach. This means each client will have different interfaces and based on their requirements they will use the system in multiple ways. For example there is a scenario where User Interface might communicate with a specific service to produce the specific piece of information that needs to be catered and displayed for the end-user. To understand in a better possible way let us consider an example of american e-commerce website where user requests the information of jewellery based on certain price range that varies between $500 to $1000, in this scenario the application will send the user request to web service for retrieving the data from the database that process the specific query and publish the resultant data on the User Interface based on the design. To get this information the system will communicate with the different layers.The complexity of the software is dependent on the number of layers used for designing the software. In similar manner, from the testing point of view the number of layers involved in development must also be tested simultaneously. From the aforementioned e-commerce test scenario, a tester need to perform front-end GUI testing and back-end testing. From the observations that by employing n-tiered automation framework approach the organization can have a quality product in an average of 5 hours by running 500 manual tests and by automating around 650 test cases that include functional, regression tests as to determine if there are any breaks in the code. If the sprint is accommodated for 4 weeks then the company could save noticeable man working hours.

2.2 GRAFT Approach :-

Eun Ha Kim et al[17] introduced a framework based on modularity driven tests that develops an abstraction layer for each UI component so that it can be re-used anywhere in the application. Each script in this framework constitutes various modules and methods of acceptance tests. Also these individual scripts can be combined into large scripts. Pajunen et al[14] proposed keyword driven framework in which the the manual tests are executed based on the keywords mentioned in the external file. The functional testing can be represented with a notation in the actions script. G. Little et al[8] suggested a data-driven based framework where the test data will be loaded into the variables that are captured in scripts. Here both the input and output data of each test scenario will be read from the properties and data files. E. M. Maximilien et al[9] suggested hybrid framework which is a combinational approach of keyword and data driven framework. The objective of this framework is to develop re-usable methods and achieve scalable throughout the target application. But this approach can be difficult to accomplish in keyword based framework approach. In GRAFT approach authors have broken the test scenarios into business based keywords. These keywords later were arranged in external files to form

end to end scenario based on business requirements. This approach has simplified the formation of test script process. Also to improve the re-usability, a common repository of test data will be maintained in external sheet so that test data can be re-used in various test scenarios.

2.3 GUI Pair :-

Maintenance of GUI tests is a continuous process and also considered to be a time consuming action in the regard of GUI automation. There are numerous authors who have suggested various approaches to deal with this Task. If any kind of broken are observed in GUI Scripts, Water [20] mechanism is suggested to fix the broken automation scripts. Its functionality is to record the DOM states in sequence manner for the previous iteration of data whether the status of the execution is passed or failed. Their approach is to analyse the elements location via exploration based technique that would ultimately helps in passing the test script. In addition, it will trigger the suggestions for the users to do further modifications for the script to run them successfully in the next iteration without any errors. By looking at the refactoring settings Daniel et al[3] suggested an automatic test script repairing process directly from the script refactoring. Grechanik et al [20] suggested a tool that inspect the steps prior to any impacts if any changes are submitted to UI in the application. This can be achieved by making a comparison among before and after GUI modification. The benefit of this tool is to display the impacted steps for the testers to make a review and repair accordingly. Memon et al [10] proposed a technical way for GUI repair tests by transforming the scripts based on the user specifications applied. This approach will help the tester to insert any new step for any broken script. Zhang et al [22] suggested a Flowfixer approach which is considered as an exploration-based technique for automatically migrating the scripts to the new specifications. By comparing the above said techniques, our mechanism will develop the scripts that are modification resident for a batch of modifications, the repairs can be externally by the non-technical testers who do not have any knowledge.

2.4 UI visual analyzer :-

Collins et al [4] have observed that it is futile to make investment in automating the GUI test scripts until the application UI development achieves stability. Berner et al [18] suggested a compendium based on the real time experiments conducted in UI automation. During this process they have faced major challenges, among which the significant concern is the creation and maintenance of GUI test scripts - here maintenance refers to the code change in the UI which will lead to the modifications of the existing scripts. This step is mandatory as there is slight tendency to undergo code breaks in the GUI scripts which will produce larger amount of false positives errors. This means the test script execution will be less compared to the expectation rate, which will create a question on the investment that marks a significant place in automation domain. To analyse the web pages in visual manner WebDiff [19] helps to detect if there are any differences in the UI layouts across the various browsers. To analyse the whole web applications and also to identify the functional failures across different browsers Mesbah et al [11]suggested dynamic crawling

2.5 Change Resiliency tests :-

To enhance the identification of cross-browser defects there is a combination approach known as crosscheck is suggested by choudhary et al [21]. This mechanisms are quite useful in determining if there are any functional incompatibilities and visual changes across the browsers. This approaches are quite beneficial from our proposed approach by adapting the script syntax. Eventually there are other approaches which are used for generation of test scenarios that helps in creating residence tests.

3 Methodology

To automate any web based application in different browsers and on various operating systems, metrics such as estimated time and human efforts is required to configure the aforementioned infrastructure as to perform cross-browser testing. From a start-up based company point of view, to support cross-browser testing on a wide variety of operating systems will incur huge financial costs. To overcome this problem, we have proposed

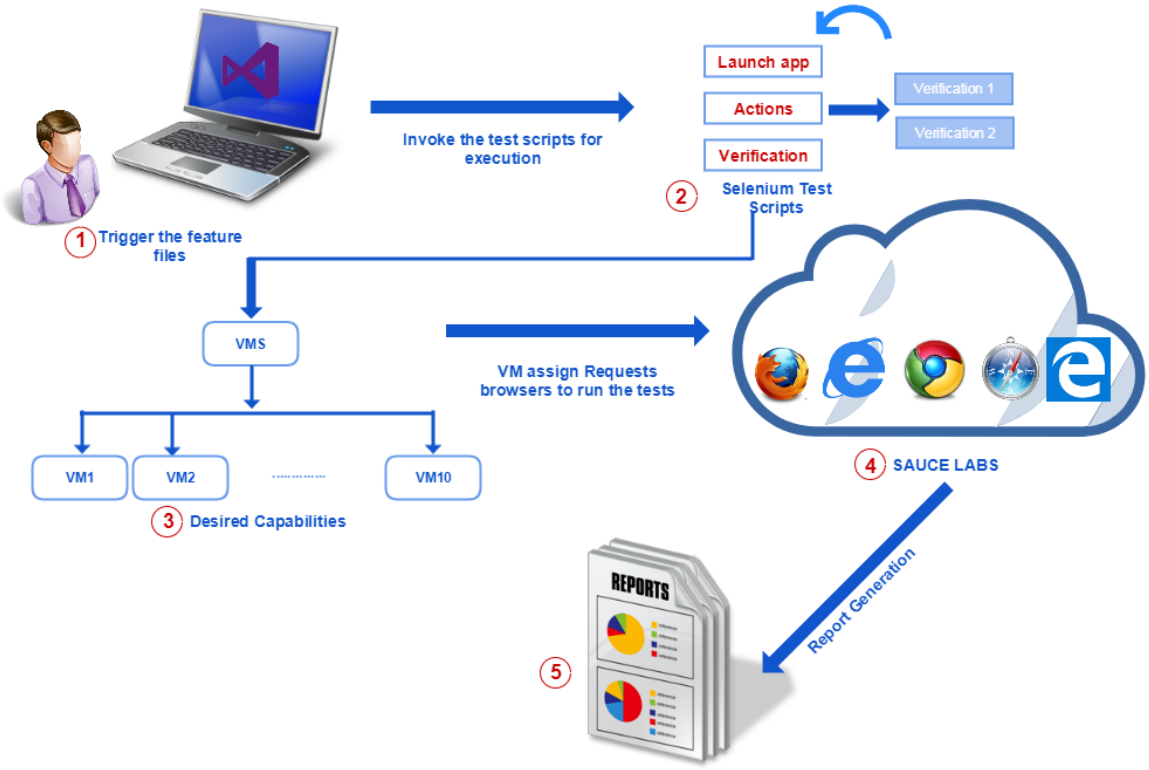


Figure 1: Sauce Lab Integration for Cloud Testing

an architecture that will target different browsers and perform regression, functional testing based on the test cases development. Here in the proposed architecture we are integrating the reusable framework with cloud-hosting testing platform known as Sauce Labs. Sauce labs will allow the user to perform the testing around on 850 browsers

that includes different versions of each browser like Firefox, Chrome, Internet Explorer, Safari and Opera etc. It also allows us to perform cross browser testing on devices that includes android based and IOS based of different screen sizes. In the current work, we are focusing on desktop based browser versions as to check the compatibility of the application is functioning as expected based on the requirement specifications.

Figure 1 depicts the overall architecture of our proposed solution and it is composed of 5 phases. They are as follows

1. Trigger the feature Files : In this phase, the test steps are mentioned in feature files which will invoke the wdio engine to call the respective functions to start the execution for each test step.

2. Selenium Test Scripts : In the second phase each test step will perform respective action by calling the function mentioned in the selenium test scripts (ex: I openUrl "www.myntra.com" will call the openURL function resided in Given Function).

3. Desired Capabilities : In the third phase, we create a desired capabilities file that holds the different browser information that are required to run the test scripts on them. Basically the intention of this file is to accommodate the VM for each O/S on the cloud-hosting platform to execute the tests in parallel or sequential

4. Sauce Labs: In the fourth phase we integrated the cloud testing platform namely Sauce Labs to invoke the VM requests on browsers mentioned in desired capabilities to run the tests. We can invoke the browsers on Sauce Labs only if sauce service is enabled in the script and the parameters referred in Listing 1

5. Report Generation: In the final phase we integrated the allure reporting feature that will fetch the test results in .json format and will convert into human readable format by using the following command "allure generate test-results". The benefit of integrating the reporting tool is to determine the count of defects and execution time

```
Set SAUCE_USERNAME= 'XXX'     // Enter UserName
Set SAUCE_ACCESS_KEY= 'XXX'   // Enter Access Key

/* services: ['sauce'],
    user: process.env.SAUCE_USERNAME,
    key: process.env.SAUCE_ACCESS_KEY,
    sauceConnect: true, */
```

Listing 1: Sauce Connect Parameters

At a time we can invoke only 5 Vm instances by setting the keyword Maxinstances as follows

.

```
Set Max_Instances = 5  // Instantiates 5 browser instances for test
   case executions
```

Listing 2: Maximum Instance parameter

The reason for using 5 instances is the limitation provided in the trail version of Sauce Labs account Invocation of the instances will enable all the browsers to start the execution of the test cases based on parallel or sequential setting simultaneously. To determine the

execution time, we are collecting the time metrics after the completion of running of each test case for each browser, as this metrics will be quite helpful in evaluating whether parallel tests or sequential tests are suitable for regression test suite. Eventually in the final phase of the architecture we are integrating with test reporting feature as it will fetch the statistics of each test step whether it is passed or failed. If any defects or failures occur during the test case execution, user will get detailed information of what went wrong in the report and it will also trigger respective error code on the console for easy debugging. In addition, memory usage is also calculated while running the test case as to detect any memory leaks while performing actions on the browsers. These statistics provide a strong support for performance testing .

Internal Architecture of Test Automation Framework:-

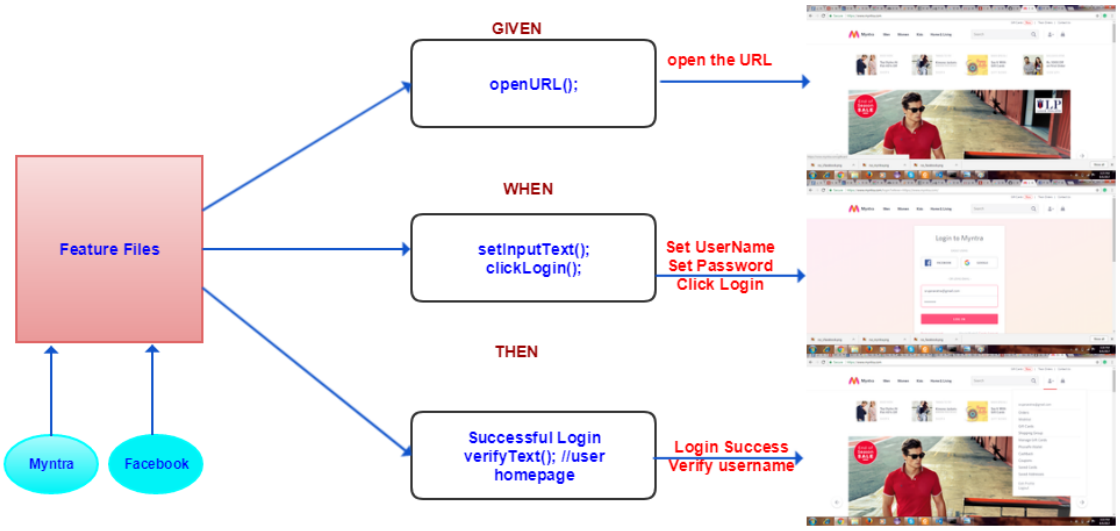


Figure 2: Internal Framework

In our approach we developed an internal test automation framework that helps a novice to understand about how test scripts are written and how each test step will call the function in the backend and how it executes on the browser. We used the guidelines of Gherkin syntax for developing the action based scripts. We segregated the actions like opening a URL , providing the data in the input fields, click on login button and eventually verifying the text if the login is successful. From the figure 2 if we observe "Given","When" & "Then" keywords are used and the objective of writing a test scenario is to split into three different sections and assign the specific actions to these keywords. Below are the keywords and their importance.

1. *Given*: This section helps to determine the pre-conditions to begin the test.
2. *When*: This section is where the tester will specify the behavior
3. *Then*: This section will describe the expected changes from the specified behavior.

```
1 Given: I open the URL "www.myntra.com"
2 When: I set the input to the field "set username"
```

```
When: I set the input to the fiels "set password"
And: I click on the "Login" button
Then: Check the "set username" is present in the home page
```

Listing 3: Feature Example

From the figure 2 the test scenario “login to the myntra application” will contain series of test steps that will perform the respective actions on the browser instance invoked by the feature file. As per the example mentioned in Listing 3 the feature example, To open a URL “www.myntra.com” will map with the openURL() function in the Given Script and perform its action on the browser. Similarly to set the input fields like UserName and Password will map with the setInputText() function in the When script, also to click on the login button it will map with the clickLogin() function. Eventually to check if the login is successful we are verifying the username in the homepage for that verifyText() function will be mapped to the respective test step in Then script. Similarly in this way we have developed nearly 366 test cases that includes different types of actions like sorting, searching, filtering, re-sizing the images. The basic idea of developing this individual action specific functions are for re-utilizing them for other web applications such as flipkart and amazon. Therefore, the tester just needs to develop the feature files and use the reusable functions.

4 Implementation

We present our solution by implementing a re-usable automation framework that allows the non-technical testers to create feature files for the target application "www.myntra.com" and run against different browsers in sauce labs.

4.1 Re-usable Feature test Automation Framework:-

To develop a re-usable automation framework, initially as a tester the primary duty is need to consider the repeatable actions that will act as part of number of test scenarios. For each test scenario there will be sequence of steps that will be mentioned in feature specific file extension.The Feature Test Automation Platform will allows the tester to authenticate the automation based test-scripts using a Gherkin specific Language. The GSL will follow english based syntax making the end-users to easily understand and maintain the test scripts.FTAP holds .feature extension which will converts the tests steps written in GSL into JavaScript using WebDriverIo API. The test script in FTAP contains .feature specific files and functions in Given, When and Then scripts. Below figure will give the syntax used for writing the test step

```
Scenario: Hover on user icon and click on LOGIN
    Given I open the url "https://www.myntra.com/"
    And   I wait for 10000 ms
    When  I move to element ".desktop-userIconsContainer"
    And   I wait for 5000 ms
    When  I click on the webelement "a[data-track='login']"
    And   I wait for 5000 ms
    Then  I verify that webelement ".login-title" matches the text "
       Login to Myntra"
```

Listing 4: Sample test scenario

From the above example, it is very clear that what actually each test step is about and now lets understand how the mapping or conversion into javascript will be done through webDriverIO API.

```
import openURL from '../Myntra_support/action/openURL';
function given() {
    this.Given(
        /^I open the (url|site) "([^"]*)?"$/,
        openURL
    );
    }
```

Listing 5: My Javascript Example

Here the Xtext and Xbase strings will be used as part of regular expression. This means at point of context searching, through the framework the tester can issue possible keywords. Ultimately the keywords should match the regular expressions. The main goal of introducing the GSL instructions in the feature files is to fulfill the following criteria

1. The GSL should be relatively match with the language that a non-technical user should understand and adopt the framework.

2. The GSL should be restricted to the most frequently utilized commands.

Below are the certain characteristics of the feature files

1. Easily understandable by non-programmers to create feature files and enables them to understand the flow of the test script so that it can be maintained without any programming knowledge.

2. Enable the testers to pick the command from the view proposals and for every selenium command specific GSL keyword need to be developed that eventually maps with the respective function in the back end code.

3. To ensure the test coverage we will develop the functions that invokes the WebDriverIo API's directly from GSL instructions.

4.2 Invoke of JavaScript coding from GSL:-

During the process of designing GSL, we should consider the most frequently used selenium commands in Gherkin Format. Incase if there are any commands that are not covered then a mechnism is required that invokes these non-represented commands via GSL to make sure that there is complete coverage

```
var boolVar = false
Scenario: First I would like to search for an item
        Given I open the url "https://www.myntra.com"
        And I update variable boolVar = JavaScript {
  console.log("Hello world inside GSL!");
  return true;
}
```

Listing 6: OpenURL function in Given Script

The Javascript can be invoked through the "JavaScript" keyword in the GSL. We have included the XBloc statement syntax of the XText Framework. From the above figure the JavaScript will return the data type and then it will be assigned "boolVar" variable mentioned in the GSL.

4.3 Assertions and Error Logging on console:-

To create tests in expressive way, we have chai assertions that runs with promise specific assertion libraries. Let us consider following example to understand how assertions can be dealt with webdriverio. In the below figure the tester is verifying a image21 in myntra application contains "Tommy Hilfiger". Since it is a e-commerce application the products will be updated on daily basis. Here using the chai assertions we are able to find out what went wrong. As per the error triggered on the console it is clear that Image21 does not contains "Tommy Hilfiger" instead it contains new updated brand "Highlander Blue Slim Fit Denim Shirt". This information is pretty enough for a non-technical tester to understand this specific error. For the next iteration of run, the tester can easily modify the changes in the feature file for "10 test case" and make the test passed.

```
10) Verify image21 I expect that element ".pdp-title" contains the text
    "Tommy H
ilfiger":
expected 'Highlander Blue Slim Fit Denim Shirt' to include 'Tommy
   Hilfiger'
running chrome
AssertionError: expected 'Highlander Blue Slim Fit Denim Shirt' to
   include 'Tomm
```

```
y Hilfiger'
    at World.module.exports (F:/project/cucumber_myntra/src/
       myntra_app_support/check/checkC
ontainsText.js:55:25)
```

Listing 7: Assertion error code snippet

For error logging, let us consider a test case where the target web element is not located in the application. In this case the webdriverio make use of the PROMISES to trigger the error on the console. From the error it is very clear that the target element is not present in the page hence this test caused failure.

```
5) Login using Facebook7 I move to element ".desktop-userIconsContainer
   ":
An element could not be located on the page using the given search
   parameters.
running chrome
Error: An element could not be located on the page using the given
   search parameters.
```

Listing 8: Element not found error

4.4 Integration of automation testing with Sauce Labs

Saucelabs is a cloud-enabled cross browser and platform testing that facilitates the testers to run their automation scripts and manual scripts on wide variety of browsers installed on various operating systems and device emulators. This approach of cloud testing does not require software company to configure any infrastructure setup like installing multiple vm's of different versions and flavours. Currently as part of our research work we are targeting Firefox, Internet Explorer, Chrome, Microsoft Edge and Safari Browser to run the test-scripts that were developed using our proposed Feature test automatwd framework.

```
var webdriver = require('selenium-webdriver'),
    user = "thesisproject",
    key = "9662c8f4-0665-4ec0-8ec6-9cb19845a3bd",
    browser;

browser = new webdriver.Builder().
  withCapabilities({
    'browserType': 'chrome Browser',
    'operatingSystem': 'Windows 7',
    'versioNumber': '51.0',
    'userName': user,
    'accessKey': key
  }).
  usingServer("http://" + userName + ":" + accessKey +
              "@ondemand.saucelabs.com:80/wd/hub").
  build();

browser.get("http://www.myntra.com");

browser.getTitle().then(function (page_title) {
    console.log("Page Title is: " + page_title);
});

browser.quit();
```

Listing 9: Code for Connecting To Sauce Labs

4.5 Metrics of Execution time, Cpu Usage and Memory Usage

After completion of executing every test case we are collecting the execution time, cpu usage and memory usage statistics. The importance of this metrics is to check on which browser the execution time is more and also this is an efficient approach for memory leakage detection. In the traditional approach, especially to perform memory leakage detection we basically use chrome browser and take the heap snapshots.Make a comparison between the acquired snapshots by repeating the test case, in this way one can identify if there are any memory leaks the usage between the two heap snapshots will be drastically high around 500 MB. Then such discrepancy can be considered as defect as part of UI memory leak testing. So if we follow the similar approach through out the application then huge manual efforts is required. To prevent this for each test case run we are collecting the memory usage as to see whether there any memory leaks.

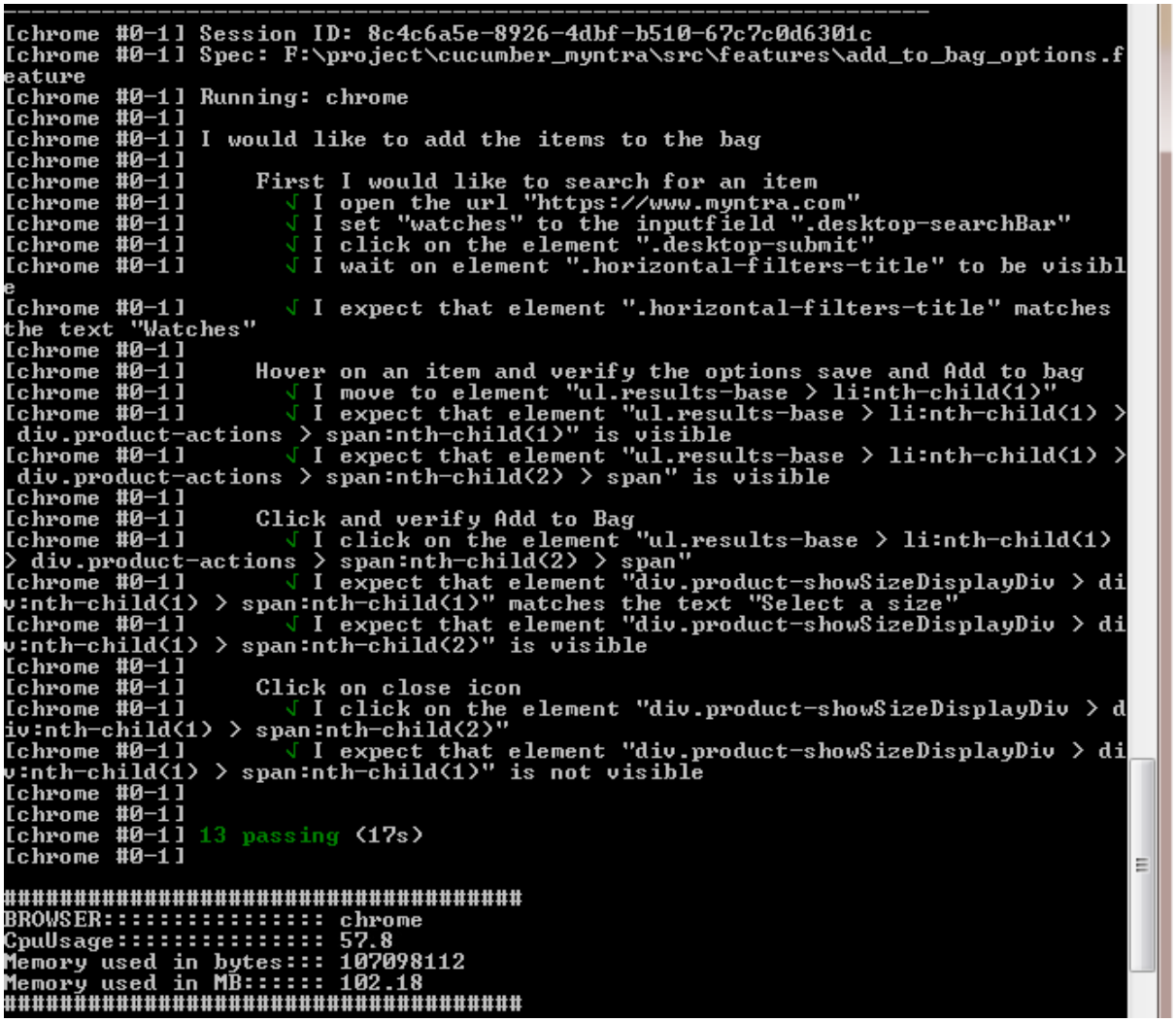


Figure 3: Memory Usage stats

5 Evaluation

In this section we will provide the results conducted for the experiments mentioned in the implementation section. In the first case study we evaluated that the proposed framework can be reusable on multiple browsers across different applications. Furthermore, we evaluated the sauce lab integration that invokes the parallel tests on three different browsers namely firefox,chrome and IE. We presented the metrics of memory usage consumption and execution time in Table 3.To record the test status after each run, a reporting feature is enabled that will provide the severity of the defects and execution time.

Using the Re-usable Feature Test Automated Framework we have developed 363 test cases that covers wide areas in www.myntra.com application. In the first case study we will show the results of re-usable framework for multiple applications. We are focusing on two different domain based web applications. They are e-commerce application "myntra" and money transfer application "Transferwise".The objective of re-usable framework is to develop feature files and run them on browser without changing back-end code.

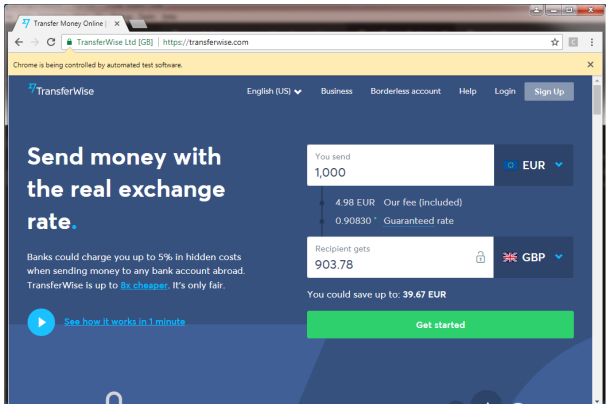
5.1 Framework Re-usability Evaluation

In this case-study we have written registration feature file for TransferWise and also executed the feature file of myntra registration file. The purpose of this two test scenarios is to check whether we are able to run the test cases without any modification to the back-end code.
We also shown the execution time taken by these two feature files.

Table 1 depicts execution of registration test case that uses same functions from the framework solution but executed against different applications.

Table 1: Registration Scenario for multiple applications

Test Case Scenario	Browser	Execution Time (secs)
Myntra Registration	Chrome	20
Transferwise Registration	Chrome	22.5



(a) Transer Wise

(b) Myntra

Figure 4: Running the tests on multiple applications

Table 2: Execution time for features

Feature File	No. Test Scenarios	Browser Execution Time in secs		
		Firefox	Chrome	IE
add_to_bag_options	13	28	30	32
add_to_checkout_complete	19	40.10	50.50	58.02
bread_crumb	8	10	20	30
contact_us	16	26.4	28.4	35
forgot_password	12	14	16	18
items_total	19	42	46.90	50
login_using_facebook	19	32.30	36.70	41.02
login_using_google	21	33.21	36.90	35.33
navigate_home	18	34.21	37.70	32.25
search	15	32.30	35.80	28.30
menu_bars	18	23.02	19.70	20
show_more_products	14	38.23	41.20	45.03
signup	17	31.03	23.30	25
sortby	13	23.60	23.90	25.8
valid_login	25	38.54	45.80	45.2

5.2 Execution of Test Cases in Sauce labs

We have integrated Sauce Labs a cloud testing platform that executes the test cases on 800 browsers of different o/s. From our research point of view we have executed the test cases of each feature file in three different browsers namely firefox, chrome and IE as these are the popularly used browsers .We have presented the execution time consumed by each feature file on different browser in sauce labs. We ran nearly 266 test scenarios In case if any test steps failed then respective error shots is also captured. To invoke the test case execution at Sauce Labs, tester needs to provide username and accesskey that will help in connecting to the O/S running on the VM.In addition, Browser name, browser version, O/s name must be mentioned in the script in order to execute the tests. Table 2 presents the different type of functional features that include "login using facebook google account" in the myntra application, add to cart , total items added in the cart, etc., that were covered during the test case execution on three browsers parallely in the Sauce Labs. This experiment is conducted as to check whether the functional tests were successfully passed on different browsers or not as part of cross-browser testing.In addition, we have collected the execution time of feature file on each browser as to evaluate the performance testing.

5.3 Metrics of Execution time and Memory usage

The main purpose of memory usage statistics is to find memory leaks while running the test scenario. This approach will not help in finding the memory leakage defects but helps in maximizing the chances for leakage detection if observed any during the test. Table 3 shows the memory consumed by each feature file and the execution time.

Table 3: Memory usage for test scenarios

Test Case Scenario	Execution Time in secs on chrome	Memory Usage in MB
add_to_bag_options	30	96.4
add_to_checkout_complete	50.50	77.14
bread_crumb	20	100.86
contact_us	28.4	99.37
forgot_password	16	101.57
items_total	46.90	60.02
login_using_facebook	36.70	60.35
login_using_google	36.90	57.48
navigate_home	37.70	75.69
search	35.80	102.39
menu_bars	19.70	102.72
show_more_products	41.20	54.9
signup	23.30	100.43
sortby	23.90	101.84
valid_login	45.80	60.8

5.4 Test Reports

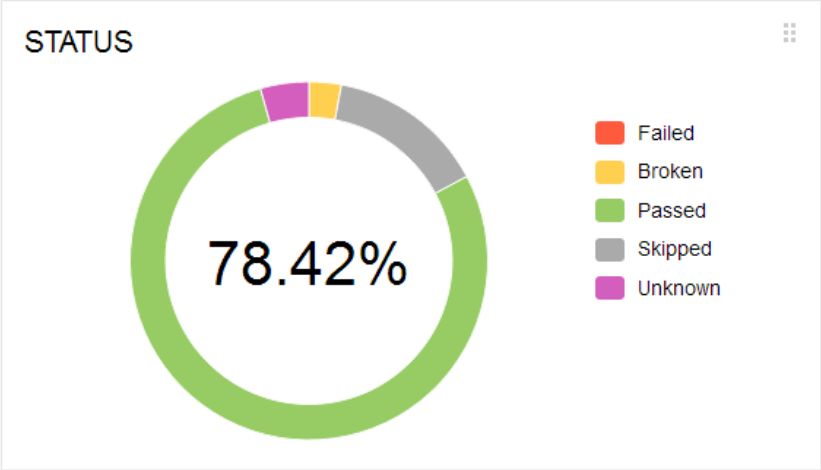
We have integrated a test reporting tool that will collects the information in json format initially. By using allure reports command it will convert the json format into html format.

In the Figure 5a, it is displayed that 78.42% of tests have been executed and the test results are segregated into five parts in pie-chart representation they are 1) Failed 2) Broken 3) Passed 4) Skipped and 5) Unknown. This way of representation will help us know how many defects have occurred during the test execution. So that they can be raised in the defect management tool for tracking purpose. Those statistics also helps whether the defects are fixed for the next iteration of the execution run. To know the statistics of execution time taken by each test scenario during the execution we have Figure 5b that will represent no.of test scenarios that were executed in seconds. On the x-axis if we consider the time frame between 3 secs and 5 secs nearly 22 test case scenarios have been executed.The majority of the test cases were taken 1 second to complete the execution. To know the severity level of the defects we can get the details as shown in in the 5c.

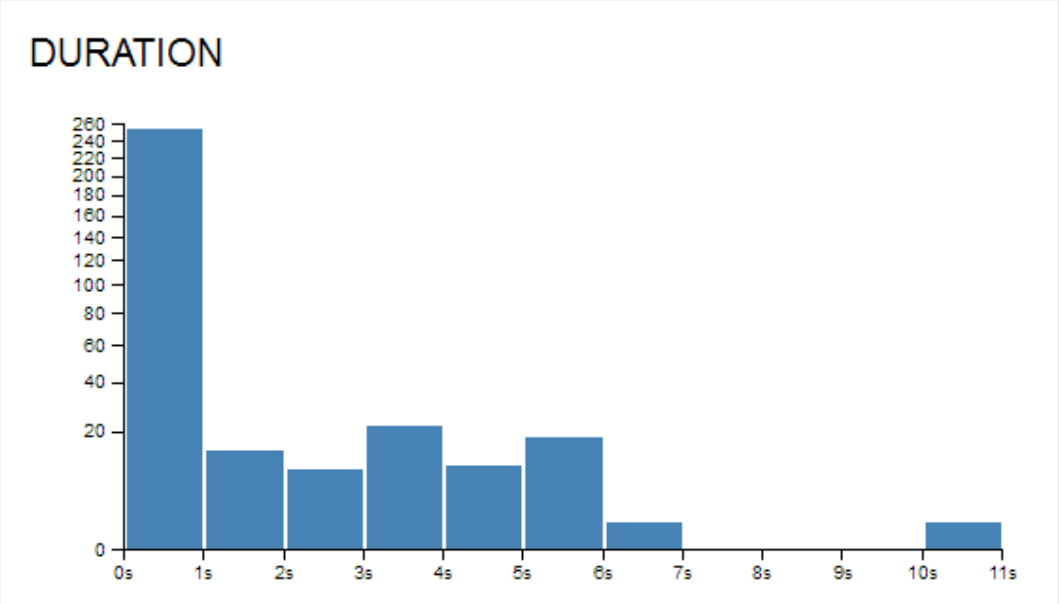
5.5 Discussion

In our research we performed various experiments by considering target application "www.myntra.com" test scenarios by running them on local machine presented in Table 3 and on cloud presented in Table 2 . We have also showed the demo of two application's test scenarios that can automated using the designed re-usable framework without any modification in the back-end. We have considered nearly 12 feature files that comprises of around 266 test cases that were ran against different browsers on cloud. During the execution of each test we have collected the memory stats, execution times along with test step's status. The benefit of using this re-usable feature test automated frameowrk

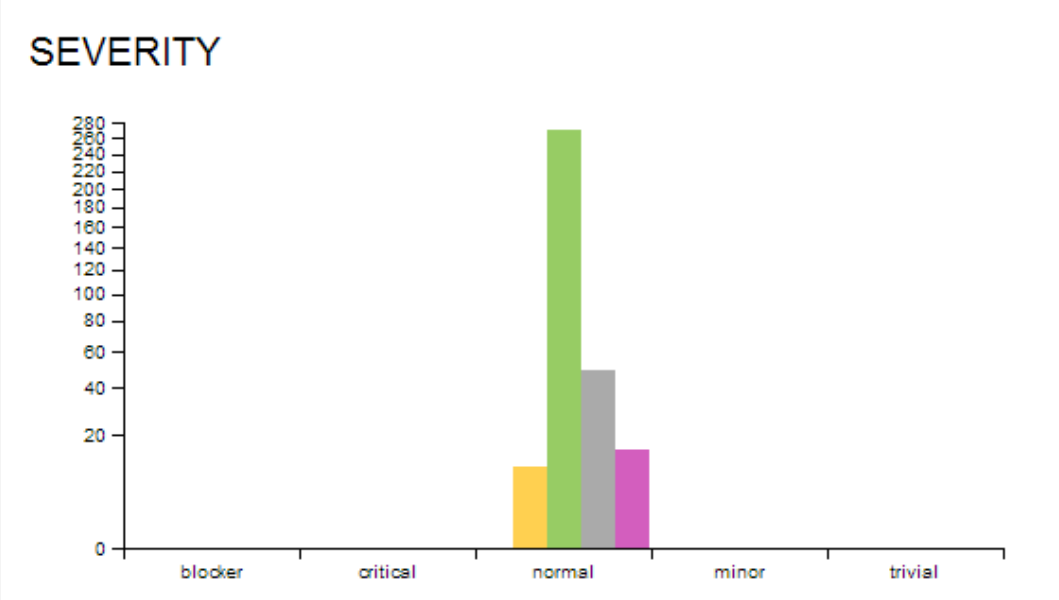
Figure 5: Test Report features



(a) Defects



(b) Time duration



(c) Severity of defects

is to automate multiple applications by adapting it in *Given*, *When* & *Then* format. In case if there are any failures in the steps it will notify the required actions like assertions which can be handled without any programming knowledge. We have integrated test reporting tool that will collect the information after completion each test run in the json format. All this json files will be collectively converted into readable format during the test report generation. From the test report stats we can view how many defects are found along with the severity level. One important aspect of this test reporting feature is it gives information of failure test steps for further correction. So in the next iteration tester need to make sure that defects were resolved. This framework is highly suitable for developing functional tests and run them for every sprint as part of regression tests in agile methodology.

From the test report case study it is observed that using the re-usable feature test framework we have executed 366 test scenario's in 10 minutes span. If we try to run them manually then it is known fact that a large number of human working hours is required. For every sprint in agile methodology the tester is responsible to check/verify the previous build functionality is successfully working fine with the newly introduced features. Therefore, the automation plays an important role and the framework which we have designed is easily adaptable, understandable, maintainable by non-technical tester.

6 Conclusion and Future Work

We have proposed a reusable framework which can be supported for multiple applications just by developing the feature files. This framework is highly suitable for running regression tests in agile methodology as it can be easily maintainable, adaptable by non-technical testers who do not have any sort of coding language. The Gherkin language instructions in the features are well presented in such away that any layman can understand the test scenario. To add an extra benefit of this framework we have integrated with the Sauce Labs which allows a tester to run the automation framework in a wide variety of browsers. Moreover, we have collected the memory stats after completion of each test to help in detecting the memory leaks. In future, we would like to extend this work by adding support to Android and IOS versions of mobiles for testing any application using our framework.

Acknowledgements

I would like to extend my sincerest thanks to my supervisor Dr. Adriana Chis for her continuous support and motivation during this research. Her confidence in my research proposal has boosted me up in exploring the new innovative ideas until the final stage of shaping up the solution. Her timely advice and guidance have benefited me in producing the thesis with all the questions answered in well organized way.

References

A. M. de Castro, G. A. Macedo, E. F. C. and Dias-Neto, A. C. (2013). Extension of selenium rc tool to perform automated testing with databases in web applications,

Vol. 5 of *in proc. of Automation of IEEE 8th International Workshop on Software Test (AST)*, IEEE, San Francisco, CA, pp. 125 – 131.

Atif Farid Mohammad, H. M. (2011). Cloud services testing: An understanding, Vol. 5 of *Procedia Computer Science*, Science Direct, pp. 513 – 520.

B. Daniel, Q. Luo, M. M. D. D. D. M. and Pezze, M. (2011). Automated gui refactoring and test script repair, In Proceedings of the First International Workshop on End-to-End Test Script Engineering, pp. 38 – 41.

Collins, E. F. and de Lucena, V. F. (2012). Software test automation practices in agile development environment: An industry experience report, Vol. 5 of *In Proceedings of the 7th International Workshop on Automation of Software Test*, IEEE, pp. 57 – 63.

Holmes, A. and Kellogg, M. (2006). Automating functional tests using selenium, *in proc. of IEEE Agile Conference* pp. 6–10.

Leotta, M. (2013). Capture-replay vs. programmable web testing: An empirical assessment during test case evolution, *20th Working Conference on Reverse Engineering (WCRE)* .

Leotta, M. (2014). Visual vs. dom-based web locators: An empirical study, *International Conference on Web Engineering. Springer International Publishing* .

Littler, G. and Millery, R. C. (2006). Translation of keywords into exec code, *In Proc.. of the 19th ACM Symposium on User Interface Software and Technology* pp. 135–144.

Maximilien, E. M. and Williams, L. (2003). Assessing test-driven development at ibm. in proceedings of the 25th international conference on software engineering, *IEEE Computer Society* pp. 564–569.

Memon, A. M. (2008). Water: Web application test repair, Vol. 18 of *ACM Transaction Software Engineering Methodology*, pp. 1 – 4.

Mesbah, A. and Prasad, M. R. (2011). Automated cross-browser compatibility testing, In Proceedings of the 33rd International Conference on Software Engineering, pp. 561 – 570.

Micallef, M. and Colombo, C. (2015). Lessons learnt from using dsls for automated software testing, Vol. 8 of *Software Testing, 2015 IEEE Eighth International Conference on*, IEEE, pp. 23 – 28.

Mugridge, R. and Cunningham, W. (2005). Fit for developing software: framework for integrated tests, *Pearson Education* .

Pajunen, T.; Takala, T. K. M. (2011). Model-based testing with a general purpose keyword-driven test automation framework, *in Software Testing, Verification and Validation Workshops (ICSTW)* pp. 242–251.

Persson, C. and Yilmazturk, N. (2004). Establishment of automated regression testing at abb: industrial experience report on avoiding the pitfalls, Vol. 8 of *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, IEEE, pp. 23 – 28.

Rankin, C. (2002). The software testing automation framework, Vol. 41, IBM Systems Journal, pp. 126–139.

Ryoo, E. H. K. J. C. N. S. M. (2009). Implementing an effective test automation framework, Vol. 2 of *in Proceedings of COMPSAC 33rd Annual IEEE International*, IEEE, pp. 534–538.

S. Berner, R. W. and Keller, R. K. (2005). Observations and lessons learned from automated testing, Vol. 5 of *In Proceedings of the 27th international conference on Software engineering*, IEEE, pp. 571 – 579.

S. Choudhary, H. V. and Orso, A. (2010). Webdiff: Automated identification of cross-browser issues in web applications, In IEEE International Conference on Software Maintenance, pp. 1 – 10.

S. R. Choudhary, D. Zhao, H. V. and Orso, A. (2011). Water: Web application test repair, Vol. 5 of *In Proceedings of the First International Workshop on End-to-End Test Script Engineering*, pp. 24 – 29.

S. R. Choudhary, M. R. P. and Orso, A. (2012). Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications, In Proceedings of the International Conference on Software Testing, Verification and Validation, pp. 171 – 180.

S. Zhang, H. L. and Ernst, M. D. (2013). Automatically repairing broken workflows for evolving gui applications, In Proceedings of the International Symposium on Software Testing and Analysis, pp. 45 – 55.

Stejskal, R. and Siy, H. (2013). Test-driven learning in high school computer science, *in proc. of IEEE 26th Conference on Software Engineering Education and Training (CSEET)* pp. 289–293.

T. Lalwani, S. N. Kanoujia, T. H. and Smith, M. (2011). Quicktest professional unplugged, *KnowlegeInbox* .

Thummalapenta, S. (2012). Automating test automation, Vol. 2 of *34th International Conference on Software Engineering (ICSE) on*, IEEE, pp. 26 – 32.

Wei-Tek Tsai, Y. H. and Shao, Q. (2011). Testing the scalability of saas applications, Scalability of SaaS applications. In Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA '11), IEEE Computer Society, Washington, DC, USA, pp. 1 – 4.

Yalla, M. and Shanbhag, M. (2009). Building automation framework around open source technologies, Vol. 10 of *in proc. of Software Testing Conference*, IEEE, pp. 6–9.

A First Appendix Section

Below are the
feature file format
GivenStatement script Format
ThenStatement Script Format
WhenStatement Script Format
Memory stats code

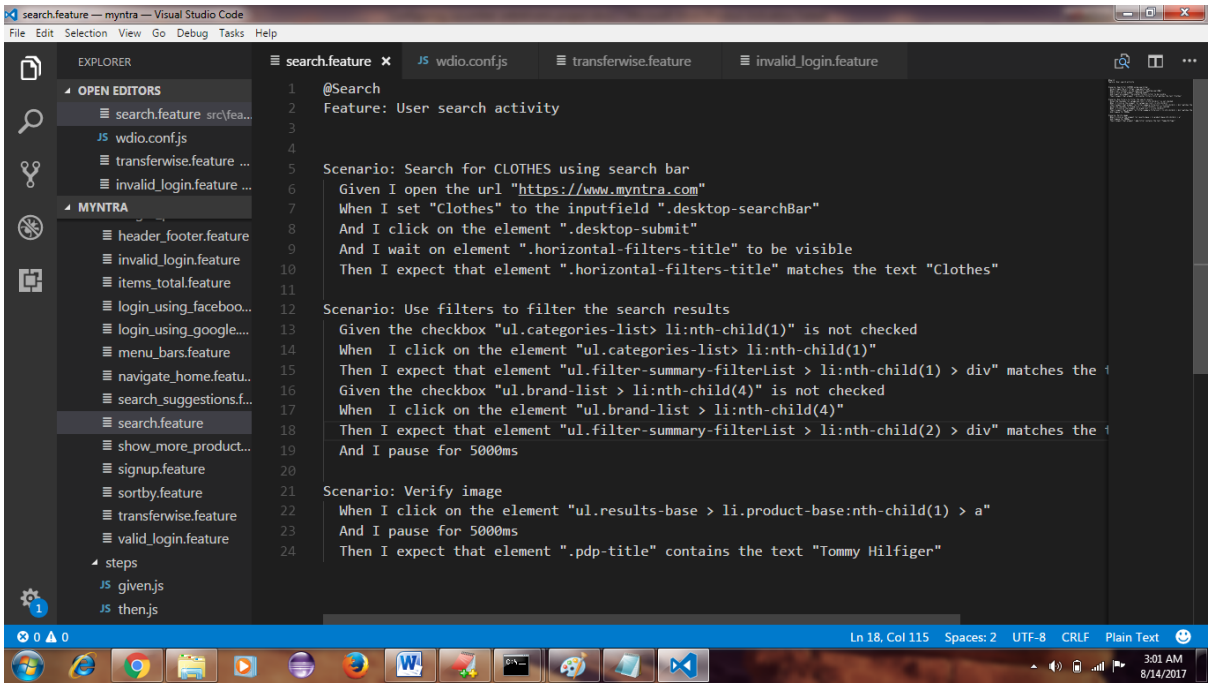


Figure 6: Feature File Format

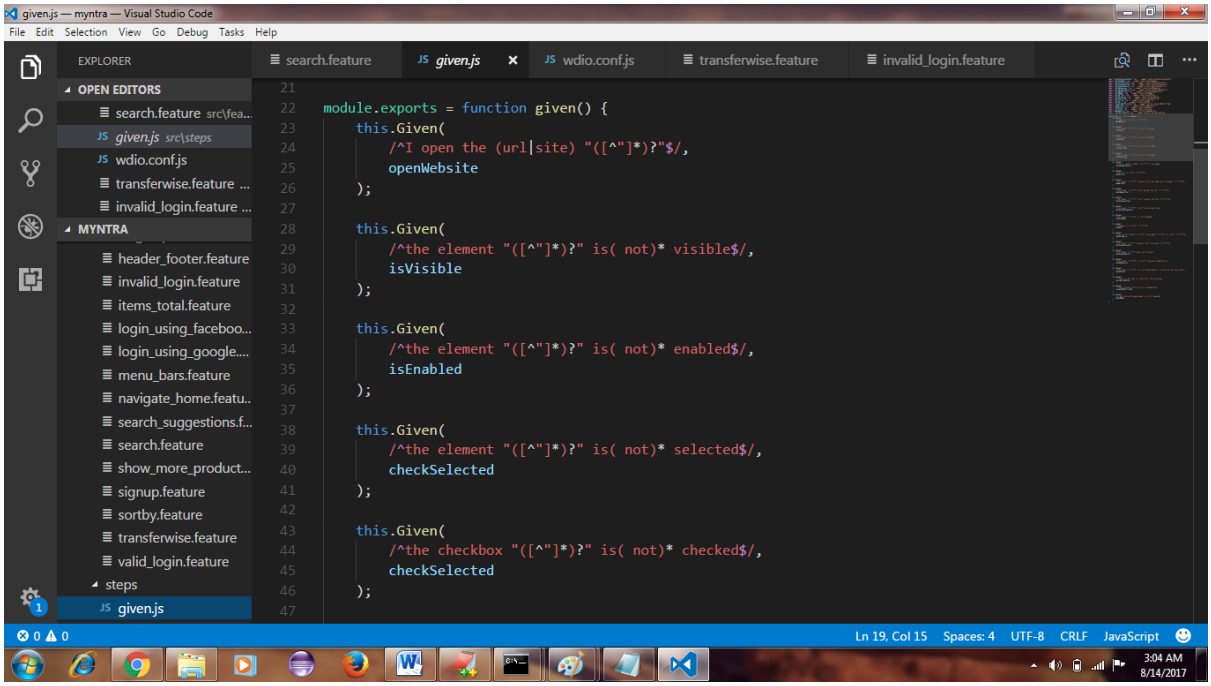


Figure 7: Given Script

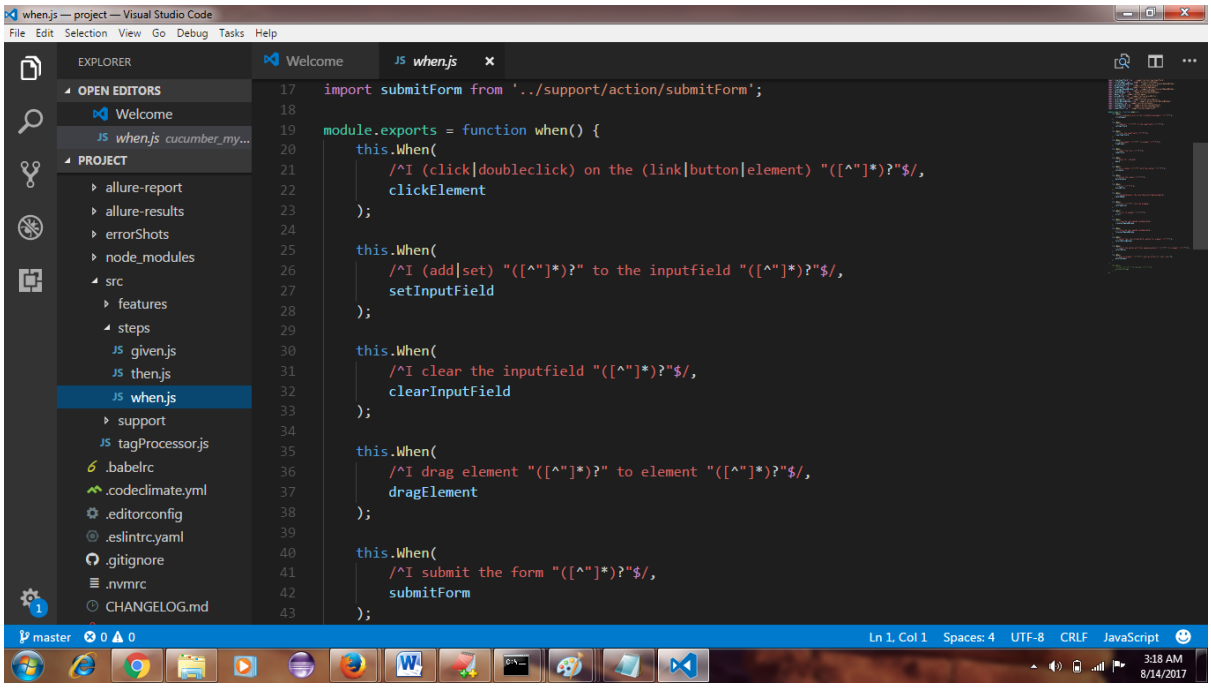


Figure 8: When Script

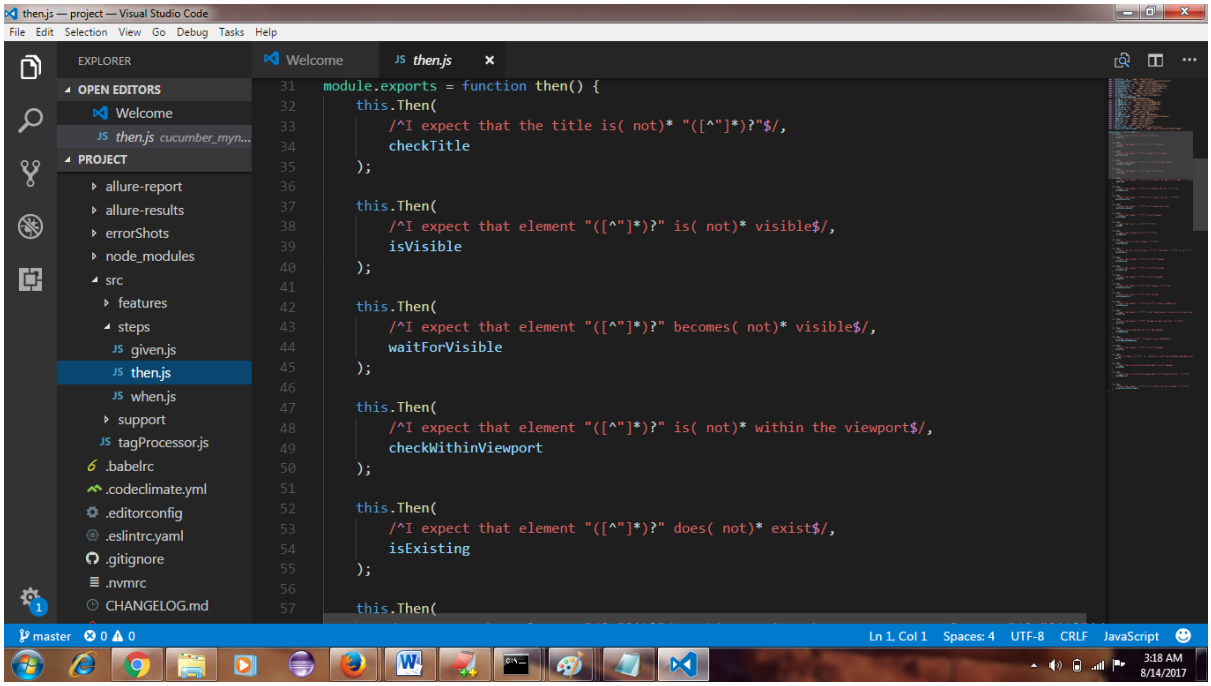


Figure 9: Then Script

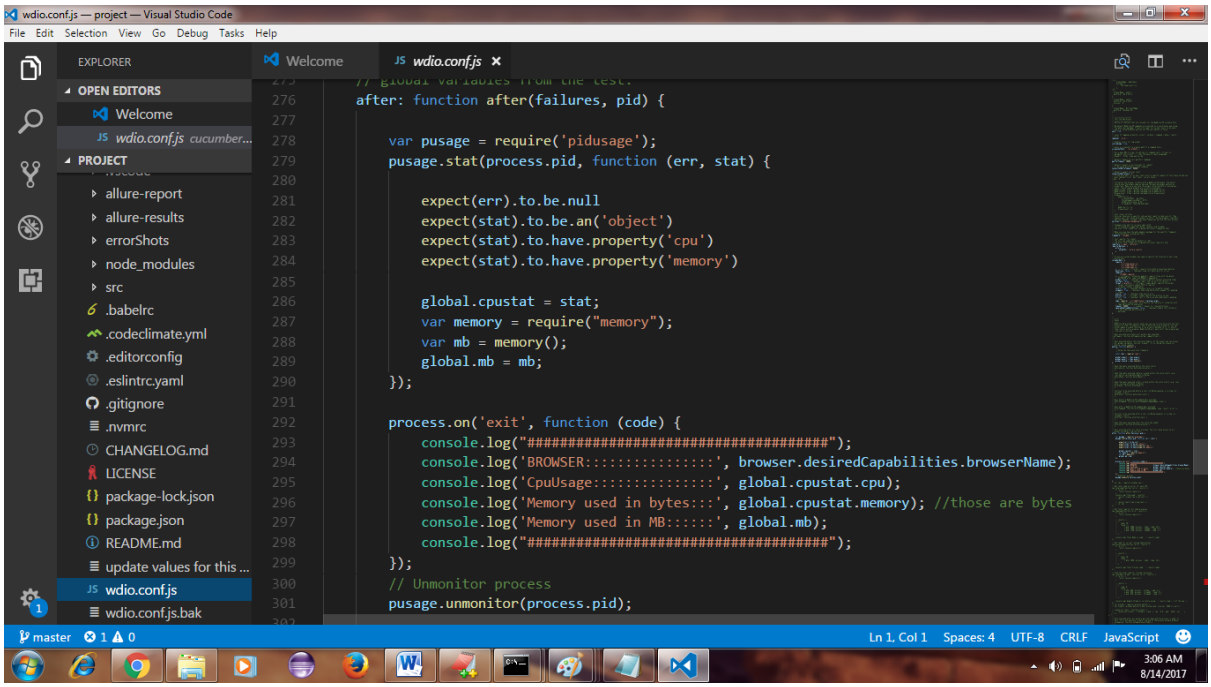


Figure 10: Memory stats code

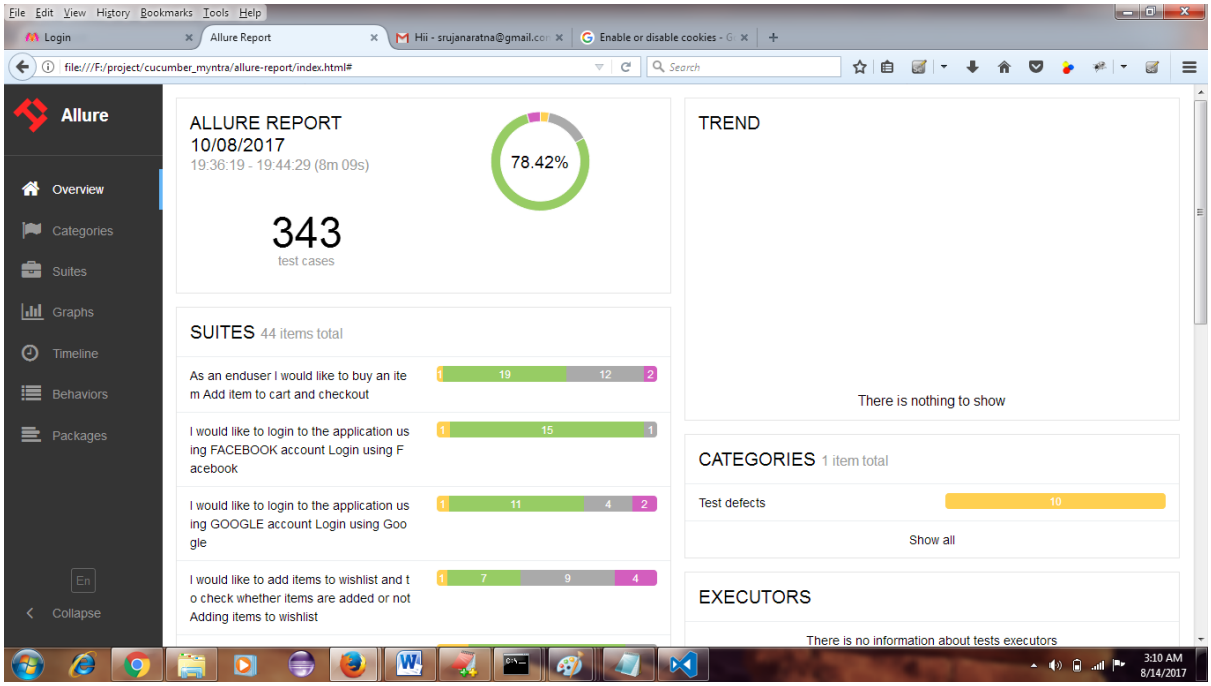


Figure 11: Reports Stats

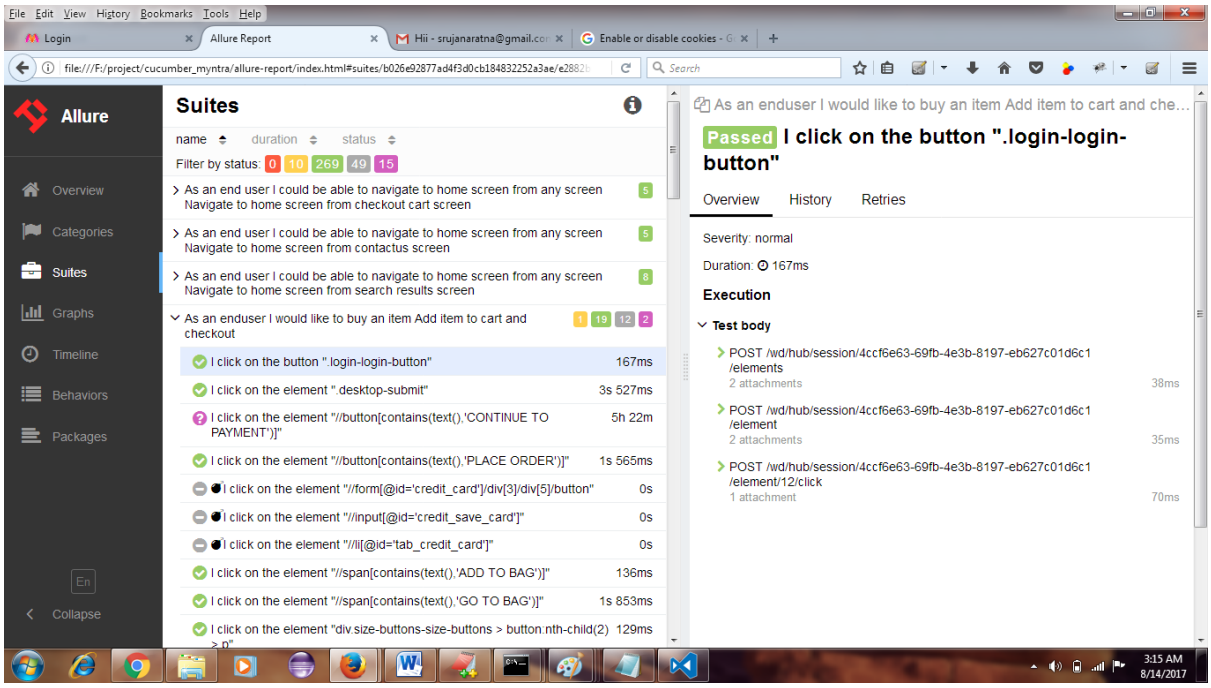


Figure 12: Test steps information