AUTOMATION REGRESSION TESTING FOR SAS.AM WEBSITE

Harutyun Berberyan and Shahid Ali

Department of Information Technology, AGI Institute, Auckland, New Zealand

ABSTRACT

This research study is focused on a company which operated in online shopping. The company entered into the online market without proper testing. The company's site was migrated from local server to Amazon Web Services which required additional changes in its site architecture. Having automation testing especially in this case, regression test suite needs to be applied for the mentioned changes. It will be very useful for quickly testing the functionality of the site and further to validate that everything is working as expected. In order to conduct the mentioned regression testing through the test automation Selenium Webdriver was selected as a test automation tool/framework and TestNG framework was added to the test automation environment to generate comprehensive reports. After test execution the results showed that first of all the automation testing is more than 3 times faster than manual and human interaction is led to the minimum. Moreover, it proves that the core functionalities were not suffered from architectural changes although some minor bugs have been revealed during the collective execution of test cases. This research will create the regression ready solution on sas.am testers' and developers' hands also it will be a good test automation framework for all web applications created on 1C-Bitrix framework, which is getting popularity.

KEYWORDS

Amazon Web Service, Application Programming Interface, Page Object Model

1. INTRODUCTION

Nowadays number of web-based applications are deployed in different platforms and ongoing trend is to make upgrade, code modification or migration of those applications from one platform to another. In those situations, automation testing is used to perform testing by reducing the human intervention and repeatable tasks. The regression testing is one step ahead in automation testing to reveal the faults that could be increased as a result of new changes in the system.

This research study is focused on automation testing which is going to be performed on the company's website. From privacy reasons, the company name will not be disclosed. The company is located in Yerevan, Armenia and they have a supermarket chain in the same city. Today the company has a turnover of greater than $3.5 million and 800 employees. In order to gain more profit, they operate online shopping web site "sas.am" which has entered into the market without decent testing. The website was created on 1C-Bitrix framework and supports three different languages and currencies in order to target English, Armenian and Russian speaking client's market. Also, the website is integrated with a call centre which operates 24 hours each day in order to handle clients' requests. The company sells a wide assortment of food, sweet, beverages and household products which can be ordered through the "sas.am" web site and delivered within the city by some additional cost. Brief objectives of this research are mentioned below:

Natarajan Meghanathan et al. (Eds) : CSEIT, CMLA, NeTCOM, CIoT, SPM, NCS, WiMoNe, Graph-hoc - 2019
pp. 117-137, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91310

- Perform website regression testing via automation testing.
- Create test automation environment utilizing Selenium WebDriver as a tool/framework for "sas.am" website.
- Integrate TestNG framework with Selenium WebDriver to simplify the "sas.am" website testing processes and generate a proper report about executed test cases.
- Create maintainable and reusable test cases using functional-decomposition approach and Page Object Modelling.
- Create a ready testing solution on testers and developers' side to easily check the functionality as expected and fix the bugs.
- Generate and track the quality metrics for continuous improvements in product quality.
- Identify criteria for selection of functionality and write test automation script to verify and validate the following functionalities: change of site language from Armenian to English, change of currency from AMD to USD, place order of bread, rice, seafood, a search of product, sign in and sign out.

This research paper is organized as follow: Section 2 focuses on the literature review of the automation regression testing. Section 3 is focused on the research methodology for this research. Section 4 explains execution results for this research. Section 5 provides the discussion on the results of this research. In section 6 recommendations for future researches are provided. Finally, in section 7 conclusion to the research is provided.

2. LITERATURE REVIEW

The main reason for migrating "sas.am" website to AWS [1] environment was the flexibility and reliability provided by the cloud environment. The migration was supported by the fact that there are already 200 million PHP based active sites on various cloud platforms (Voda, 2014). Therefore, being PHP and MySQL based technology, the 1C-Bitrix is not an exception. However, besides the mentioned technologies 1C-Bitrix is using other components (e.g. jQuery, jQuery UI, CloudFlare) which makes architectural changes at the application level more complex. Those changes are performed in the database system, front-end layer and API (Application Programming Interface) layer (Voda, 2014). All the mentioned modifications lead to the need for regression testing which will be performed on "sas.am" website. Although lots of research are done on migrating existing PHP web sites from traditional hosting to the cloud, test automation frameworks and processes there is a lack of research about consequences related to the migration of 1C-Bitrix from VMWARE server to AWS. Also, regression testing results and practices are missing for those kinds of projects. Therefore, in order to create test automation environment and to develop regression testing for this research the literature review was conducted which is given below.

According to the research papers, where the comparison of three test automation tools was done, the most popular used tools are Selenium and QTP. However, QTP is not preferable because its license is very expensive. Although commercial versions provide full support which is not available in open-source tools, the latter has its advantage thanks to programmers who continuously add enhancements in open-source tools free of charge [9]. In addition to this research, an additional literature research has been done which proved that the most comprehensive and cost-effective open-source test automation tool is a Selenium WebDriver [7], [21].

Continuing the literature review and scaling up the research field, some literature review has been done in the past regarding test automation projects based on Selenium WebDriver. In the

mentioned research Selenium WebDriver was used in conjunction with Junit, TestNG and POM (Page Object Model). In one research test cases are manually implemented using Java programming language and integrating Selenium WebDriver instructions with JUnit or TestNG assertions [14]. In another research is mentioned that TestNG is developed to overcome JUnit framework's some limitations. TestNG provides new features that makes it more useful than JUnit. TestNG covers all types of testing such as functional, unit and integration [7]. Despite all advantages, Selenium WebDriver does not have any built-in features to generate the test results. In order to eliminate this limitation, the TestNG framework is used with Selenium WebDriver. Eventually, in order to have more structured, optimized and reusable scripts, there is a well-known solution like Page Object Model (POM). One of POM deployment projects was done in a small Italian company (eXact learning solutions S.p.A.). The investigation has revealed the tangible benefits of applying the Page Object Model which was used in conjunction with Selenium WebDriver [14]. Although the project was done for the testing of the learning content management domain, the practice is possible to apply across many commercial projects like "sas.am".

In order to organize the mentioned testing activities, the Scrum methodology will be applied [18]. However, before adopting the mentioned methodology the following research will try to briefly cover most popular methodologies in the software development industry. Although the Waterfall Model has proven ineffective and upcoming trend in software development is the Scrum framework the Waterfall development is still widely used in software development companies [2]. Ericsson AB located in Sweden revealed the problems in the waterfall model and made the conclusion that the utilisation of waterfall model is not acceptable in large-scale projects and where the requirements are changing often. Therefore, the company changed the development model to an incremental and Agile methodology in 2005 [19]. Agile Scrum provides the speed and flexibility in product development and having this regression test research study with a short development cycle the Scrum methodology will be proposed as a solution. After summarizing these researches, it's clear that there is no evident research that highlights regression testing of the migrated 1C-Bitrix application to AWS cloud. Hence this research will be focused on regression testing of that kind of application.

3. RESEARCH METHODOLOGY

The research execution steps for this research are given below.

3.1. Functional Automation Test Plan

The functional automation test plan for "sas.am" research is shown in Table 1. The Table 1 covers the resource planning, time estimation and environment creation aspects of the research. Windows 10 was used for the installation of Google Chrome browse, Eclipse IDE, Selenium WebDriver and Java Development Kit. Those are minimal required tools for the test automation process.

Table 1. Functional automation test plan

Functional automation test plan	
Test Environment	
Operating system platform used for "sas.am" test automation	Windows 10 64 bit
Web/database server	"sas.am" is created on 1C-Bitrix framework and located in Amazon Cloud
Web browser	Google Chrome version 76.0.3
Test automation tool/framework	Selenium, version 3.14
Additional frameworks and libraries	TestNG, testng-metrics.jar
IDE	Eclipse, version 4.11
Java Development Kit	JDK version 12
Testing scope/type	Automation/regression, functional
Test resources	
Number of testers	1
Estimated start date	23.08.2019
Estimated end date	20.09.2019
Testing hours per day	6 hours
Total testing hours	150 hours

The test planning phases for "sas.am" research study is shown in Figure 1. The Figure 1 shows that the planning process starts with analysing "sas.am" website functionalities and what type of hardware and software platforms are needed for test automation. Then the suitable candidate is selected, and trainings are organized if needed. In this research, the suitable candidate is Test Automation Engineer Intern. In this phase also the test automation tool is selected for the research which is Selenium WebDriver. In schedule and estimation tester's effort was planned for Sprint 0, Sprint 1 and Sprint 2 according to "sas.am" research. Test environment planning phase defines how the test environment is set up and who is in charge of those processes and in this case, Test Automation Engineer Intern has performed all installation and configuration tasks. The last phase describes test execution and closure of the research.

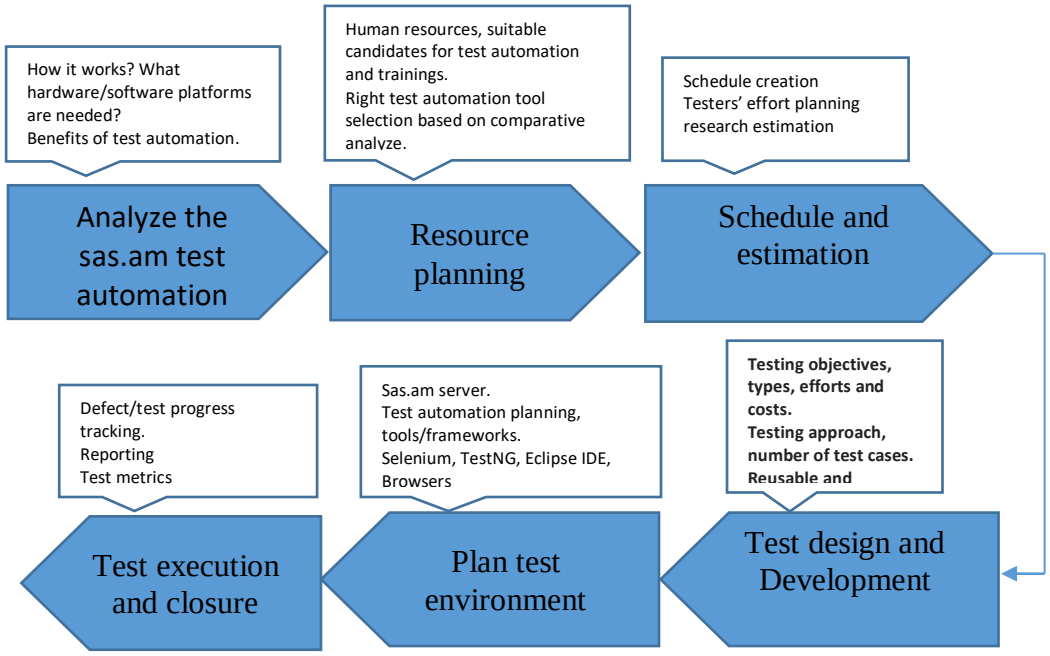

Figure 1. Planning phases

3.2.Proposed test automation framework

Proposed test automation framework for this research is shown in Figure 2, which describes the process flow and interaction between Page Factory classes, TestNG classes, Selenium WebDriver and web browsers [7]. The Page Factory class is the farther improvement to the Page Object design pattern. It is used to initialize and instantiate the elements of the Page Object. Page Factory is an inbuilt Page Object Model (POM) concept for Selenium Web Driver, but it is much optimized [14]. TestNG is integrated with Eclipse in the proposed framework in order to generate reports and to have multiple test case execution.

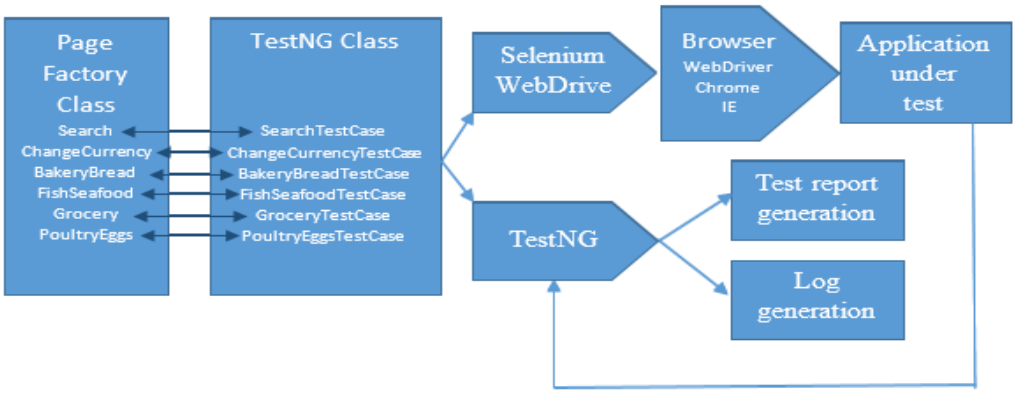

Figure 2. Proposed test automation framework

According to proposal in order to improve the maintainability and reusability of test cases, the functional-decomposition approach and Page Object Modelling (POM) is applied in this research. In the diagram the test cases are represented as Java classes. The architecture of class interaction and test execution process is shown in Figure 3.

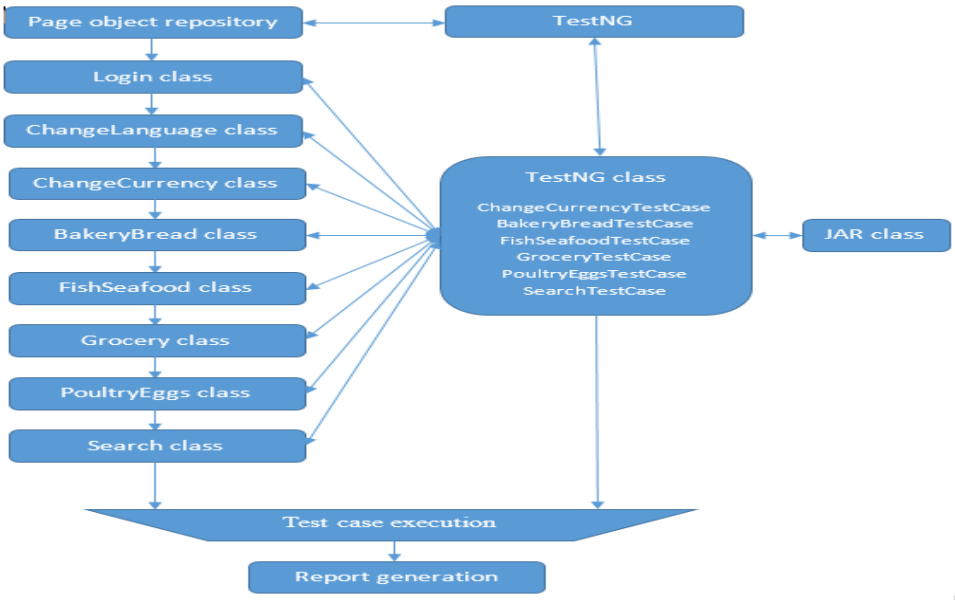

Figure 3. Architecture of class interaction, test generation and reporting

As per the POM, for every web page, the separate class has been created. Those classes have been organized in page_factory_objects package. Then another two packages have been created for the test suite and utility class which is shown in Figure 4.

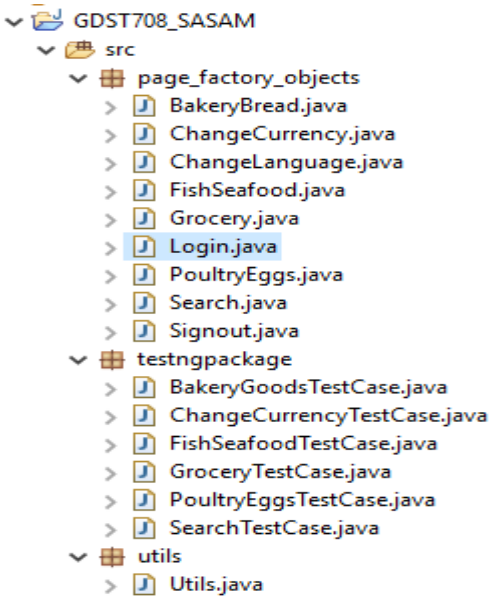

Figure 4. Organization of classes in Eclipse environment

3.3. Functional Test Cases

All selected requirements which must be done during Sprint 0, Sprint 1 and Sprint 2 are given in Table 2. Actually, the requirement is an expected behaviour of software which must be fulfilled during the testing. Thus four requirements are specified in Table 2 among those requirements are the customer registration, product order and search functionalities.

Table 2: Requirements

Req. ID	Description
Req.1	The customers shall change language and currency in “sas.am” website
Req.2	The customer shall register/sign into the “sas.am” website
Req.3	The customer shall make order of product
Req.4	The customer shall perform a search of the product

Based on functional requirements described in Table 2, the following user stories (US) are created for “sas.am” test automation research which is shown in Table 3. For example from Req.2, the following user stories have been derived:

- US3: As a customer, I want to register/login to the system so that I can add, view or change my orders

- US4: As a customer, I want to sign out from the system so that I make sure that my account is protected from other users

Table 3. User stories

User story ID	Description
US1	As a customer, I want to change the site language between three supported languages so that I can do my activities
US2	As a customer, I want to change the site currency between three supported currencies so that I can do an order of product
US3	As a customer, I want to register/login to the system so that I can add, view or change my orders
US4	As a customer, I want to sign out from the site so that I make sure that no one else can use my account
US5	As a customer, I want to make an order, view my cart or empty my cart so that I can buy an appropriate product
US6	As a customer, I want to do a search for needed product so that I can easily make an order of product

The Requirement Traceability Matrix (e.g. RTM) in Table 4 shows the mapping of user requirement with test cases. The RTM is very important because test coverage against requirements can be identified. For example, the Req.1 has been mapped to TC1 and TC2 test cases which validate the site language change functionality from Armenian to English and change currency functionality from AMD to USD. The same logic is applied on the rest requirements and test cases.

Table 4. Traceability matrix

Req. ID	Scenario ID	Test case ID	Test case description
Req.1	US1	TC1	Validate site language change functionality from Armenian to English
	US2	TC2	Validate currency change functionality from AMD to USD
Req.2	US3	TC3	Validate customer registration functionality
		TC4	Validate customer log in functionality with correct username and password
	US4	TC5	Validate customer sign out functionality
Req.3	US5	TC6	Validate order bread functionality under White Bread sub menu
		TC7	Validate order fish functionality under Fresh Fish sub menu
		TC8	Validate order rice functionality under Rice sub menu
		TC9	Validate order chicken functionality under Hens, chicken sub menu
Req.4	US6	TC10	Validate search product functionality
		TC11	Validate filter search results by min and max prices
		TC12	Validate clear search results functionality

The test case prioritization is needed because there is no lots of time and system resources to spend on full regression testing, therefore there is a need to identify which test cases should be run during Sprint 0, Sprint 1 and Sprint 2 of duration 5 weeks.

Customer registration and login functionality which are described in US3 and US4 have been given highest priority because these are the Minimum Viability Product functionalities. To avoid hips of registered user accounts into the database at present instance trying to stick with the

manual registration process instead of automation. The payment functionality has been tested manually as the credit card information can't be shared through the test automation script. The next high priority was given to order product functionality under US5 using four best-sold product statistics from the current database. Less priority was given to search product and change language functionalities under US1 and US2. Change language functionality belongs to cosmetics behaviour so it's given low priority. The user story prioritization is shown in table 5.

Table 5: User story prioritization

Task Name	**User Story**	**Priority**
Sprint 0	US3	Highest
	US4	High
Sprint 0	US5	High
Sprint 1	US5	Medium
Sprint 1	US6	Medium
	US1	Low
Sprint 2	US2	Low

3.4.Automation Test Scripts

In Test Class, the actual Selenium test automation script is written. Here also so-called page action is performed on Web Pages. For each page, its own test class is written and commented for better code readability. Test cases are written in @Test annotation which marks a method/class as a part of the test.

The small fragments of the code are used for explanation purposes. In "page object repository" nine page object classes were created, where two scripts are responsible for login/sign out functionality and seven scripts are responsible for functional test cases. As change language and login functionalities were performed during each test case, they were included in @BeforeTest annotation and the Sign out functionality was included in @AfterTest annotation. In the "testngpackage" six classes have been created where the TestNG classes (e.g. BakeryGoodsTestCase, ChangeCurrencyTestCase etc.) are controlling the test executions and the creation of an instance of Google Chrome driver, ChangeLanguage, BakeryBread, FishSeafood and for the rest classes. One of those TestNG classes is shown in Figure 5. Then TestNG Classes are controlled by testng.xml file as shown in Figure 6.

BakeryGoodsTestCase.java

```
public class BakeryGoodsTestCase {

    String Url = "https://www.sas.am";  //link of the sas.am web server
    String driverPath = "C:\\Eclipse_workspace\\Webdrivers\\Chrome\\chromedriver.exe";  //the location of the web driver
    WebDriver driver;

    //declaration of the object
    ChangeLanguage objChangeLanguage;
    Login objLogin;
    BakeryBread objBakeryBread;
    Signout objSignout;
    Utils objUtils;

    @BeforeSuite
    public void setURL() {
    System.setProperty("webdriver.chrome.driver", driverPath);
    driver = new ChromeDriver();
    driver.get(Url);
    driver.manage().window().maximize();
    System.out.println("launching chrome browser");
                        }

    @BeforeSuite
    public void testchangelanguagethenlogin() throws InterruptedException {
    //Create ChangeLanguage Page Object
    objChangeLanguage = new ChangeLanguage(driver); //creating an object by calling the ChangeLanguage class's constructor.
    objChangeLanguage.changelanguage();
    //Create Login Page object
    objLogin = new Login(driver);   //creating an object by calling the Login class's constructor.
    objLogin.logintosasam("harutyun19833@yahoo.com", "Abcd1234#"); //, "Welcome, "
                                                    }
```

Figure 5. Bakery goods test case class

*testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <classes>
        <class name="testngpackage.BakeryGoodsTestCase"/>
        <class name="testngpackage.ChangeCurrencyTestCase"/>
        <class name="testngpackage.FishSeafoodTestCase"/>
        <class name="testngpackage.GroceryTestCase"/>
        <class name="testngpackage.PoultryEggsTestCase"/>
        <class name="testngpackage.SearchTestCase"/>
    </classes>
  </test>
</suite>
```

Figure 6. TestNG xml file

In figure 7 the BakeryBread class script is depicted. The script is divided into three logical sections described below.

- Creation of the class for each functional test and finding the web elements (Figure 7)
- Performing actions on elements, like click, move to element (Figure 8)
- Creation of public method with parameters inside the class (Figure 9)

The same logic is applied on the rest of seven classes ChangeCurrency, CaseLanguage, FishSeafood, Grocery, Poultry, Search, Login and Sign out in Figure 4.

BakeryBread.java

```
public class BakeryBread {

    WebDriver driver;

    //"Bakery Goods" link
    @FindBy(xpath="//*[@href='/categories/Bakery_Goods_Buns_and_Rolls_1045/']") //web element identification by @FindBy annotation
    WebElement linkBakeryGoods;

    //"White Bread" link
    @FindBy(xpath="//*[@href='/categories/White_1064/']")
    WebElement linkWhiteBread;

    //"Bread Matnaqash" link
    @FindBy(xpath="//*[@href='/products/Bread_Matnaqash_g__370846/']")
    WebElement linkMatnaqash;

    //"Add" link
    @FindBy(xpath="//*[@class='btn' and @value='Add']")
    WebElement linkAdd;

    //"View full cart" link
    @FindBy(xpath="//*[@href='/personal/cart/' and @class='light_btn']")
    WebElement linkViewFullCart;

    //"Empty cart"
    @FindBy(xpath="//*[@class='solid ico_clear']")
    WebElement linkEmptyCart;
```

Figure 7: BakeryBread class script's part one

BakeryBread.java

```
    public BakeryBread(WebDriver driver)
    {
    this.driver = driver;   //refers to the constructor's parameter
    PageFactory.initElements(driver, this); //static init elements of pagefactory class for initializing web element
    }

    //open "Bakery Goods, Bans and Rolls" sub menu
    public void clkBakery(){
    Actions actions = new Actions(driver);
    actions.moveToElement(linkBakeryGoods);
    actions.build().perform();
                            }

    //Click on White Bread
    public void clkWhiteBread(){
    linkWhiteBread.click();
                                }

    //Do assertion 1 "I should be presented with a screen where I can select white breads"
    public void doAssertion(String expRes){
    String actRes = header.getText();
    System.out.println("Assertion White Breads: " + actRes);
    Assert.assertEquals(actRes, expRes);
    }

    //Click on Matnaqash Bread
    public void clkMatnaqashBread(){
    linkMatnaqash.click();
                                }
```

Figure 8. BakeryBread class script's part two

BakeryBread.java

```
103
104     //Creation of public method "createorderbread" with parameters String ExpRes .....
105     public void createorderbread(String ExpRes, String ExpRes2) throws InterruptedException{
106
107     //"Bakery" link click
108     Utils.wait_four_sec();
109     this.clkBakery();
110
111     //"White bread" link click
112     Utils.wait_four_sec();
113     this.clkWhiteBread();
114
115     //Do assertion 1
116     Utils.wait_one_sec();
117     this.doAssertion(ExpRes);
118
119     //"Matnaqash bread" link click
120     Utils.wait_four_sec();
121     this.clkMatnaqashBread();
122
123     //Do assertion 2
124     Utils.wait_one_sec();
125     this.doAssertion2(ExpRes2);
126
127     //"Add item to cart" link click
128     Utils.wait_two_sec();
129     this.clkAdd();
130
131     //"View full cart" link click
132     Utils.wait_four_sec();
133     this.clkViewFullCart();
```

Figure 9. BakeryBread class script's part three

3.5. Executable Jar File

A Java archive (e.g. Jar) is a process of collecting all the necessary files of the "sas.am" website test research together. The main purpose of creating this file is to distribute the single executable file of sas.am test research. The script of the executable jar file is given in Figure 10. The script contains "ExecutableJarFile" public class with instance of "jarcollector" for BakeryGoodsTestCase, ChangeCurrencyTestCase etc.

ExecutableJarFile.java

```
1  package testngpackage;
2
3  import org.testng.TestNG;
4
5  public class ExecutableJarFile {
6      static TestNG jarcollector;
7      public static void main(String[] args) {
8          jarcollector = new TestNG();
9          jarcollector.setTestClasses(new Class[] {
10             BakeryGoodsTestCase.class,
11             ChangeCurrencyTestCase.class,
12             FishSeafoodTestCase.class,
13             GroceryTestCase.class,
14             PoultryEggsTestCase.class,
15             SearchTestCase.class
16                                         });
17
18         jarcollector.run();
19                                     }
20
21 }
```

Figure 10. Executable jar file creation script

4. RESULTS

The Eclipse console logs are shown in Figure 11 with six passed and zero failed results. In that log, all the assertions are shown for change currency and different product order functionalities. The report also shows the file name where the generated metrics are stored and in this execution, it is Metrics-2019Sep13-1825.html file. In these logs the ChromeDriver version can be identified which is useful for debugging purposes.

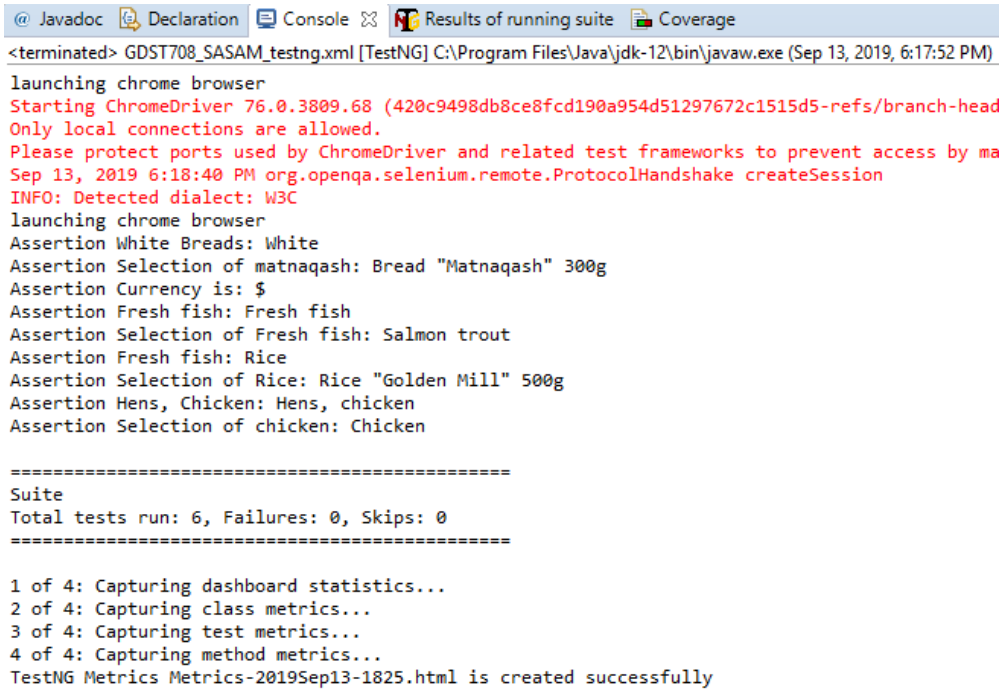

Figure 11. Console report

The precise execution time of each test case and the total execution time of the test suite are given in Figure 12. As the previous figure the Figure 12 also shows how many test cases are passed, failed or skipped. The “testngpackage” in this figure represents the package where all test classes were created.

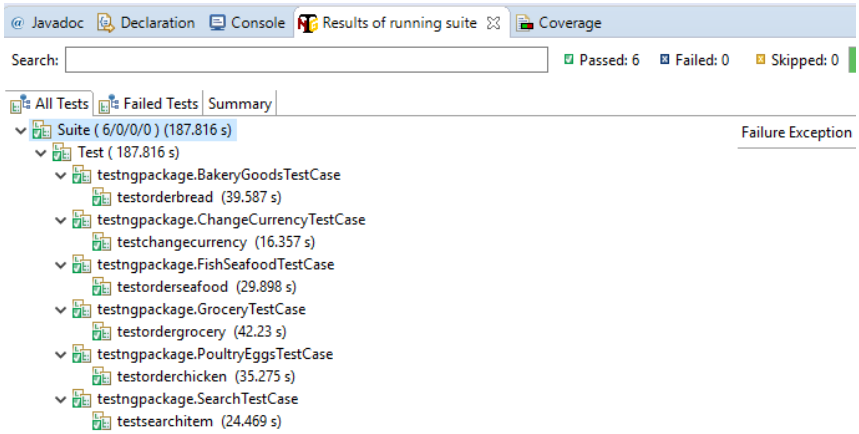

Figure 12. Results of running test suite

From the literature review we can conclude that Selenium WebDriver does not have built-in feature to generate the test results. In the framework proposed in this research TestNG is integrated with Eclipse in order to create the test report and execute test cases. This report contains all the passed and failed test cases of TestNG.

TestNG logs for timing reports of test suite are depicted in Figure 13. Actually this report has the same results as the Figure 12.

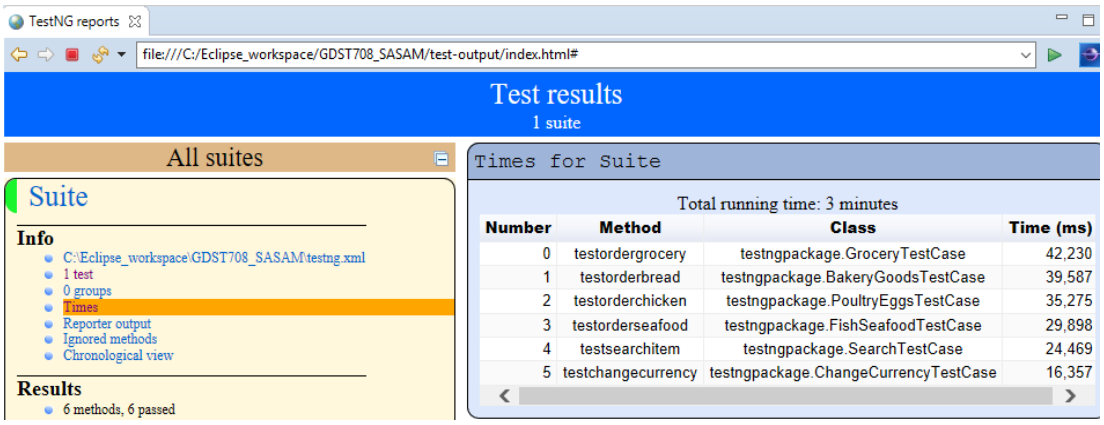

Figure 13. Timing reports for the test suite

However TestNG reports are very tedious to understand, so the "testng-metrics.jar" lib was downloaded from maven.org website and integrated into the Eclipse environment. During the execution of test cases, the "Metrics-2019Sep13-1825.html" file is generated which contains reports shown in Figure 14 to Figure 18. Figure 14 shows that six test cases have been passed and there is no failed or skipped test cases in this execution.

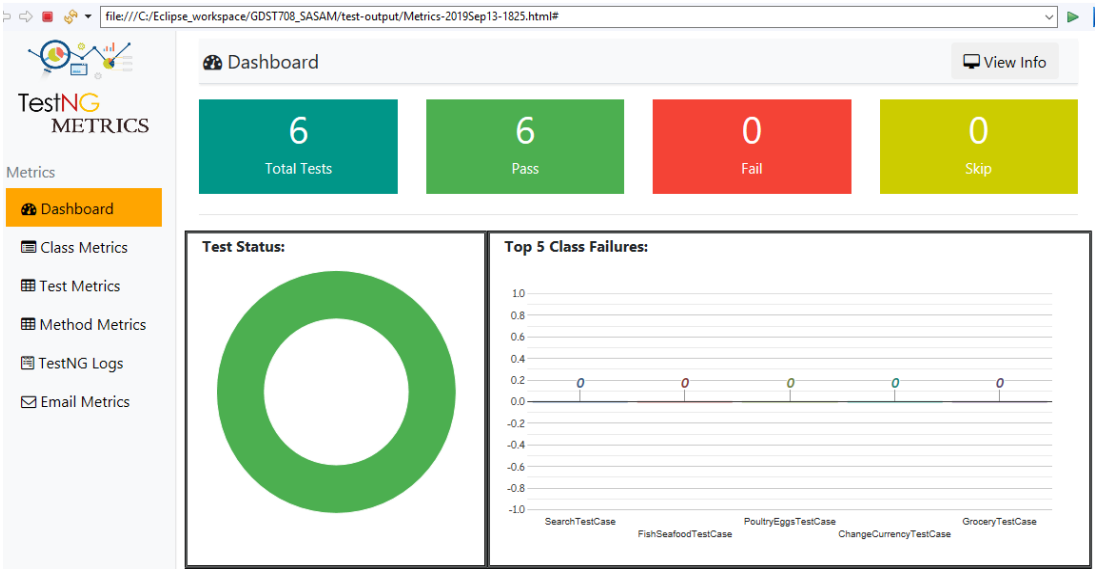

Figure 14. TestNT Dashboard report

The top ten test performances are shown in Figure 15 where the longest period of time took “testordergrocery” test execution and the shortest period of time was spent by “testchangecurrency” test execution. As mentioned, change language, login and sign out functionalities were included to all test cases except “testchangecurrency” test case, that’s why the latter is showing the smallest duration of test execution.

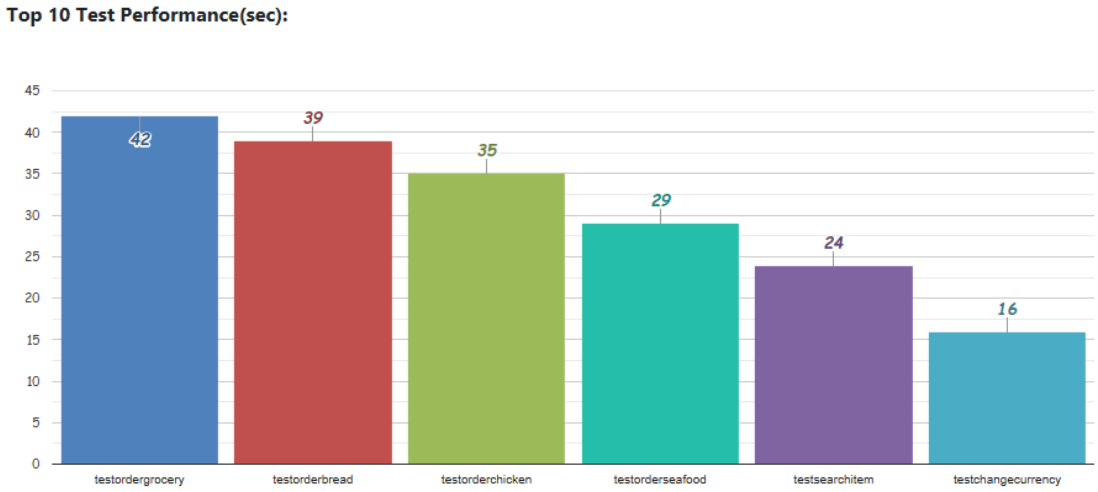

Figure 15. Top 10 Test Performances in seconds

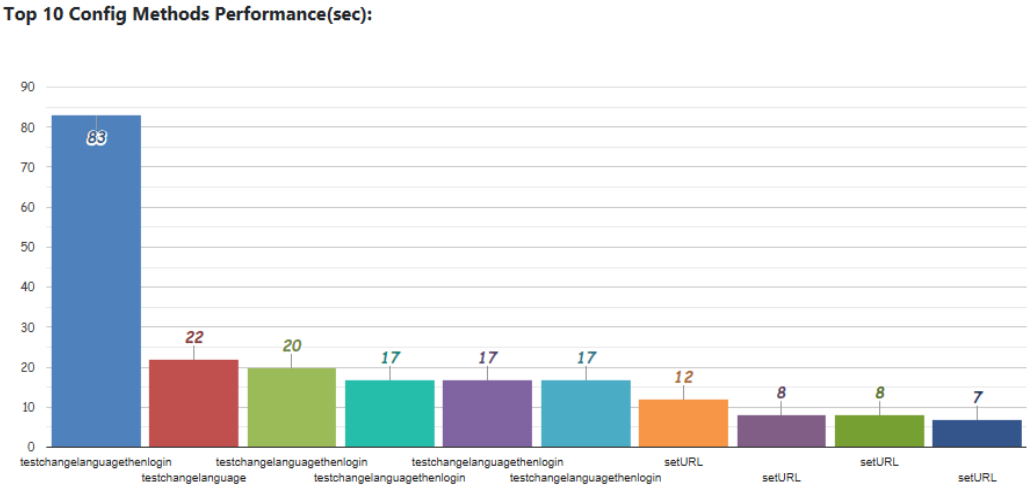

Figure 16. Top 10 Config Methods Performances in seconds

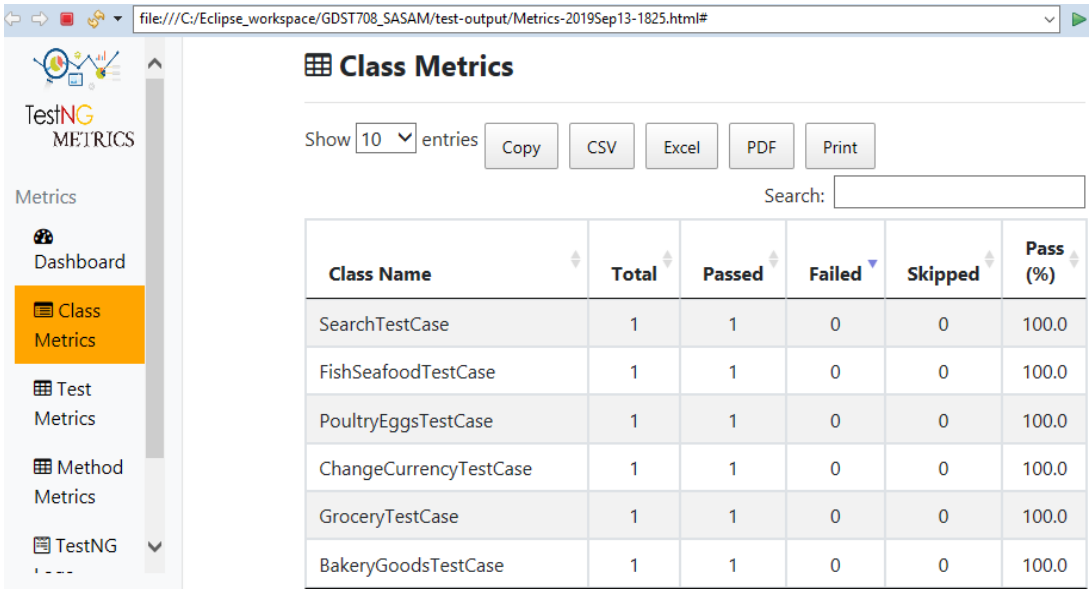

Figure 17. Class metrics

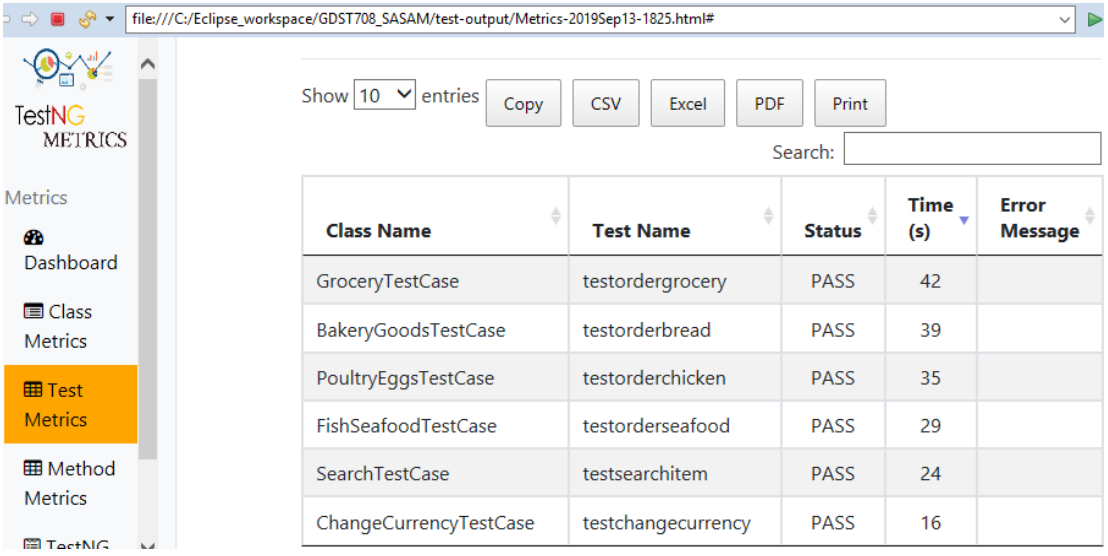

Figure 18. Test metrics

Although this report covers test automation research of “sas.am” website, the manual testing also has been performed and test execution times have been recorded for comparison purposes in Table 6. Here also change language, login and sign out functionalities have been included in test case in order to make a realistic comparison of the results between manual and automated test executions.

Table 6. Manual test execution durations of sas.am research

TestCase	Manual Testing (in seconds)
BakeryGoodsTestCase	82
ChangeCurrencyTestCase	38
FishSeafoodTestCase	63
GroceryTestCase	79
PoultryEggsTestCase	91
SearchTestCase	87
Total	440

Manual test execution screenshot is shown in Figure 19. In this screenshot, the site is opening in its default language which in this case is Armenian.

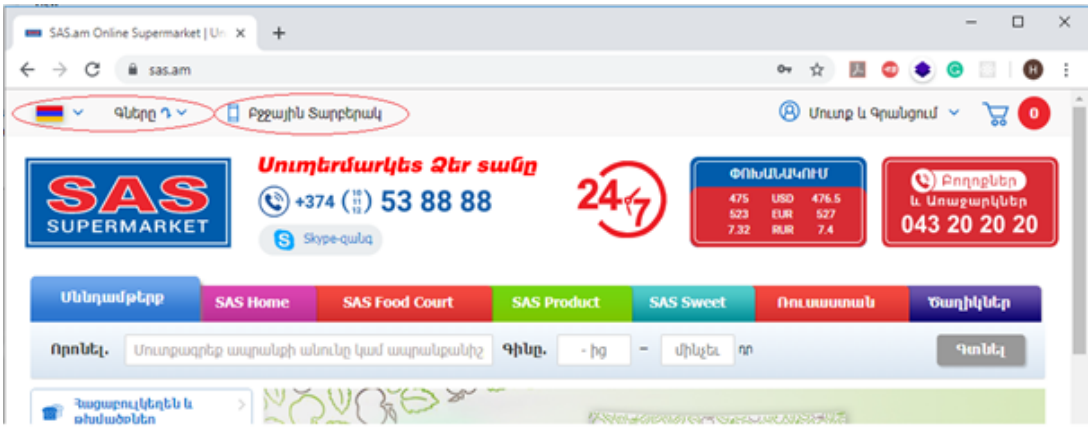

Figure 19. Screenshot taken during manual test execution

The test automation execution screenshot is shown in Figure 20, it is quite visible that Chrome web browser is being controlled by automated test software.

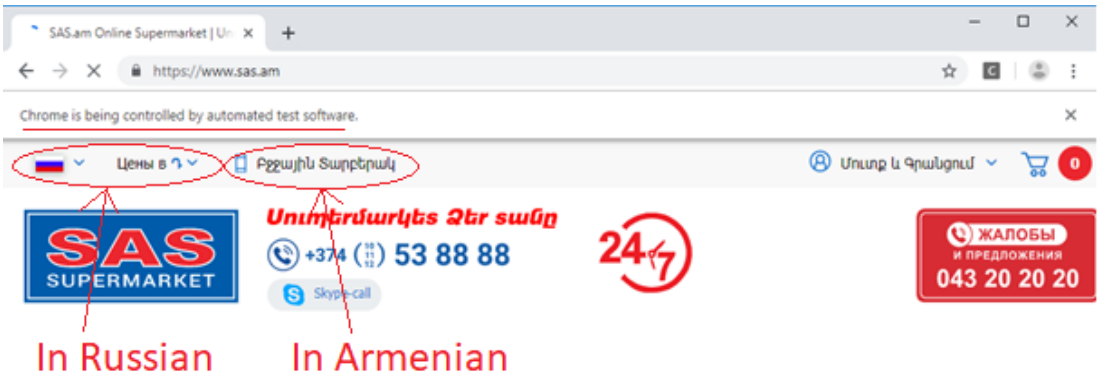

Figure 20. Screenshot taken during test automation execution

5. DISCUSSION

In this research test cases are manually implemented in Java programming language integrating Selenium WebDriver instructions with TestNG assertions.

As mentioned earlier we applied POM in this research study and Page Factory class is another form of Object Repository. Thus for each web page its own Page Object was defined**.** Each web element was uniquely identified and defined at the class level. Thus the "Find By" annotation was used and web element were defined so that actions were performed on them.

Web element identification has been done using custom XPath expressions. As most of sas.am site Web elements' "id" values were unavailable or by using only one attribute like "id" it was difficult to make the element unique therefore the combination of several attributes were used. For example, the code segment in Figure 21 shows "View Full Cart" element's identification by "href" and "call" attributes.

```
//"View full cart" link
@FindBy(xpath="//*[@href='/personal/cart/' and @class='light_btn']")
WebElement linkViewFullCart;
```

Figure 21. Identification of web element by custom XPath

For better maintainability each test case has been parameterized with different input/expected results values (e.g. String ExpRes) as shown in Figure 22 which is taken from the code fragment of "BakeryGoods" test case. This approach makes the code more optimized and reusable.

```
public void testorderbread() throws InterruptedException {
//Create "Bakery Bread" Page object
objBakeryBread = new BakeryBread(driver);   //creating an object by calling the BakeryBread class's constructor.
objBakeryBread.createorderbread("White", "Bread \"Matnaqash\" 300g");
}
```

```
//Creation of public method "createorderbread" with parameters String ExpRes .....
public void createorderbread(String ExpRes, String ExpRes2) throws InterruptedException{
```

In order to interact with the login/registration form and left navigation menu of products "moveToElement" method has been used because the "click" method was useless for those case. The code fragment is shown in Figure 23. This script imitates the mouse movement towards the left vertical menu where the "Bakery Goods" link is rendered.

```
//open "Bakery Goods, Bans and Rolls" sub menu
public void clkBakery(){
Actions actions = new Actions(driver);
actions.moveToElement(linkBakeryGoods);
actions.build().perform();
                        }
```

Figure 22. Code fragment utilising moveToElement method

Each test case of the test suite performs various steps such as navigating web pages, ordering products, filling search forms and finally performing evaluation of a set of assertions. Hence the purpose of assertion is very critical to detect issues of the product under test. As mentioned in literature reviews TestNG provides some new functionality that makes it more powerful than JUnit. Among the advantages are assertion handling techniques (such as dependent classes, Group Test, Parameterized tests etc) which TestNG provides. The sample of assertion is shown

in Figure 24 where expected result was compared with actual result which is “White", "Bread "Matnaqash" 300g”.

```
//Do assertion 1 "I should be presented with a screen where I can select white breads"
public void doAssertion(String expRes){
String actRes = header.getText();
System.out.println("Assertion White Breads: " + actRes);
Assert.assertEquals(actRes, expRes);
}
```

Figure 23. Assertion code sample from BakeryBread class

Utils.java file was created in order to store the Thread.sleep method, which pauses the execution for a specific period of time. For this research study four methods have been created to initiate a delay of execution one, two, three and four seconds accordingly. They also were used to initiate demonstrative delays for the tester during the execution. The script of Utils.java class is shown in Figure 25.

```
public class Utils {

    public static void wait_one_sec () throws InterruptedException {
           Thread.sleep(1000);
            }

    public static void wait_two_sec () throws InterruptedException {
               Thread.sleep(2000);
                }

    public static void wait_three_sec () throws InterruptedException {
           Thread.sleep(3000);
            }

    public static void wait_four_sec () throws InterruptedException {
           Thread.sleep(4000);
            }
}
```

Figure 24. Utils.java class file

The console report in Figure 11 shows test results for six passed test cases with their assertions. The results also show that there are no failed and skipped test cases and this means that the migration of “sas.am” web site to the Amazon Web Service didn’t affect those functionalities. These results are very useful to get a quick report about test execution.

Although Figure 14 shows that there are no failed test cases the visual observation during automated test execution revealed some minor bugs in user interface and evidence of it is given in Figure 20. In that image, the language flag and price is shown in Russian language and currency symbol and mobile version link were shown in Armenian language.

The results from Figure 18 show test metrics which contains the class names of the tests and their execution times. These timings are in direct ratio with the quantity of products contained in the corresponding page as shown in Table 7.

Table 7. Relation of test execution time to item numbers in the tested page

Test Case	Time in sec	Products (items) per page
BakeryGoodsTestCase	39	20
GroceryTestCase	42	20
PoultryEggsTestCase	35	6
FishSeafoodTestCase	29	2

Finally, the manual and automation test suites have been compared which results are shown in Table 8. During the automation testing, 8 seconds of the demonstrative delay was added to each test class which will be taken into consideration during the calculations of total test case execution duration. Thus, after doing the calculations the results showed that automation testing is 3.2 times faster than manual testing.

Table 8. Comparison of manual and automat test executions of sas.am research

TestCase	Test Automation (in seconds)	Manual Testing (in seconds)
BakeryGoodsTestCase	39	82
ChangeCurrencyTestCase	16	38
FishSeafoodTestCase	29	63
GroceryTestCase	42	79
PoultryEggsTestCase	35	91
SearchTestCase	24	87
Total	185-6*8=137	440

6. RECOMMENDATIONS

After conducting this research study, the recommendations are given below.

- Use Maven to easily build a project (add jars and other dependencies of the research project).
- Improve utility file by moving more methods and optimizing the existing code of sas.am test automation script.
- Execute tests parallel in AWS cloud by creating a virtual machine in the same subnetwork where the sas.am web server is located. It will help to improve the test execution performance.
- Move parametrization outside of the code and put into the CSV file. It will help to prevent direct code modification.
- Try to add more comments in the code in order to improve the readability of the script.
- Run the created regression testing on new releases of 1C-bitrix framework and compare results before making an upgrade of existing sas.am website framework.
- Put more assertions in created test scripts.

7. CONCLUSION

This report has covered test automation and regression testing framework for “sas.am” website based on Selenium WebDriver and TestNG. Although the testing and especially test automation is always recommended, the main reason for conducting regression testing was resulted because

of the recent migration of the company website to AWS platform that led to increase of the bugs in the system.

In this research study, the objectives were achieved by successfully creating an architecture of test automation framework, identification of web elements and creation of reusable automation test scripts. The research was conducted applying the Scrum methodology to expedite the testing processes. Based on prioritised user stories the test scripts were created and executed in created test automation environment. All the testing activities have been monitored and controlled by different monitoring and reporting features built into the selected test automation tool. The mentioned reporting results helped to generate and track quality metrics for continuous improvements of the product quality. Generated metrics showed testing time reduction compared with manual testing. Moreover, there were some minor bugs have been revealed by visual observation during automated test execution. The proposed framework is very significant for dynamically changing web applications like "sas.am" and it consists of reusable codes for full regression testing. Further, this research study will provide guidelines for future references regarding regression testing on migrated web applications.

REFERENCES

[1] Amazon, E. C. (2015). Amazon web services. *Available in: http://aws. amazon. com/es/ec2/(November 2012)*.

[2] Bannink, S. (2014, January). Challenges in the Transition from Waterfall to Scrum–a Casestudy at Portbase. In *20th Twente Student Conference on Information Technology*.

[3] Burd, B. (2017). *Java for dummies*. John Wiley & Sons.

[4] Davies, S., & Roper, M. (2014, September). What's in a bug report?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (p. 26). ACM.

[5] Elallaoui, M., Nafil, K., & Touahni, R. (2016, October). Automatic generation of TestNG tests cases from UML sequence diagrams in Scrum process. In *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)* (pp. 65-70). IEEE.

[6] Elbaum, S., Rothermel, G., & Penix, J. (2014, November). Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 235-245). ACM.

[7] Gojare, S., Joshi, R., & Gaigaware, D. (2015). Analysis and design of selenium webdriver automation testing framework. *Procedia Computer Science*, *50*, 341-346.

[8] Hamburger, V. (2016). *Building VMware Software-Defined Data Centers*. Packt Publishing Ltd.

[9] Hanna, M., Aboutabl, A. E., & Mostafa, M. S. M. (2018). Automated Software Testing Framework for Web Applications. *International Journal of Applied Engineering Research*, *13*(11), 9758-9767.

[10] Jain, C. R., & Kaluri, R. (2015). Design of automation scripts execution application for selenium webdriver and test NG framework. *ARPN J Eng Appl Sci*, *10*, 2440-2445.

[11] Jatain, A., & Sharma, G. (2013). A systematic review of techniques for test case prioritization. *International Journal of Computer Applications*, *68*(2), 38-42.

[12] Kakaraparthy, D. (2017). Overview and Analysis of Automated Testing Tools: Ranorex, Test Complete, Selenium.

[13] Kumar, A., & Saxena, S. (2015). Data driven testing framework using selenium WebDriver. *International Journal of Computer Applications*, *118*(18).

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

[15] Lewis, W. E. (2017). *Software testing and continuous quality improvement*. Auerbach publications.

[16] Litchmore, K. A. (2016). *A comparative study of agile methods, people factors, and process factors in relation to project success* (Doctoral dissertation, Capella University).

[17] Olsson, M. (2015). *JavaScript Quick Syntax Reference*. Apress.
[18] Permana, P. A. G. (2015). Scrum method implementation in a software development project management. *International Journal of Advanced Computer Science and Applications*, *6*(9), 198-204.
[19] Petersen, K., Wohlin, C., & Baca, D. (2009, June). The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement* (pp. 386-400). Springer, Berlin, Heidelberg.
[20] Sharma, M., & Angmo, R. (2014). Web based automation testing and tools. *International Journal of Computer Science and Information Technologies*, *5*(1), 908-912.
[21] Sheth, T., & Singh, S. K. (2015). Software Test Automation-Approach on evaluating test automation tools. *International Journal of Scientific and Research Publications*, *5*(8), 1-4.
[22] Stikkolorum, D. R., & Chaudron, M. R. (2016, July). A Workshop for Integrating UML Modelling and Agile Development in the Classroom. In *Proceedings of the Computer Science Education Research Conference 2016* (pp. 4-11). ACM.

AUTHORS

Harutyun Berberyan was born in Yerevan, Armenia, in 1983. I received my bachelor's degree in computer systems and Informatics from the State Engineering University of Armenia, Armenia, in 2005, In the same year I joined to CISCO regional academy to get CISCO instructor courses. In 2006 I started my career as a PHP and MySQL developer in "ArdNET" company. After short period I got a job offer for IT specialist position from the biggest telecom company located in Yerevan. After getting lots of experience in the company I decided to change my job and got two job offer from pharmaceutical company KrKa d.d. and SASGROUP LLC. In KrKa I was as an IT manager and in SASGROUP LLC Network Engineer. I worked in both companies since 2018 then I moved to New Zealand to study software testing.

Dr. Shahid Ali is a senior lecturer and IT programme leader of information technology at AGI Education Ltd, Auckland, New Zealand. He has published number of research papers in ensemble learning. His expertise and research interests include machine learning, data mining ensemble learning and knowledge discovery.