AUTOMATION AND PRIORITISATION TECHNIQUE FOR REGRESSION TESTING OF PB TECH WEB APPLICATION

Ho Joong Kim and Shahid Ali

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

ABSTRACT

Regression testing is a necessary process to ensure that the existing functionalities of a piece of software are not affected by new features or fixing defects. However, in the case for the web application of PB Tech, this process is very repetitive and time-consuming. In order to solve this issue, automation testing is implemented and a new test case prioritisation technique is proposed based on a combination of human- evaluation and statistical data of the highest earning features of retailer websites. Using this technique, a regression test suite is created and the test execution times are compared against a full regression test suite. The results revealed that the prioritisation technique is effective at reducing test execution times. This technique could prove to be effective for use in projects missing defect and requirements documentation.

KEYWORDS

Automation Testing, Regression Testing, Test Case Prioritization

1. INTRODUCTION

In order to add new functions to the PB Tech website or fix defects, it must be ensured that the changes made do not affect the existing functionalities of the site. Because of this, regression testing is required. Regression testing is performed to ensure that all existing functionalities are working as intended and no new defects are introduced due to changes made. However, regression testing is a very time-intensive and repetitive task when done manually, especially if all functionalities are to be tested every time there is a change made. In order to solve this problem, test cases can be automated which would significantly reduce the time and resource costs. The time and resource cost can further be reduced by prioritising the test cases so that partial regression testing can be performed instead of having to test all functionalities.

The objective of this project was to create a sample full regression test suite, that represents the functionalities of PB Tech's website, and perform test-case prioritisation and selection to form a prioritised regression test suite. The test cases for both suites were automated and comparative analysis between the prioritised regression test suite and full regression test suite was performed.
The test cases of this project were prioritised and selected using human evaluation-based test case prioritisation in addition to a newly proposed technique. Automation test scripts were written to be compiled in the prioritised regression test suite and the scripts were executed.

Natarajan Meghanathan et al. (Eds) : ACITY, AIAA, DPPR, CNDC, WIMNET, WEST, ICSS - 2019
pp. 37-52, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91704

The scope of this project was limited to the retail and customer related functionalities and did not cover the service or educational functionalities that PB Tech also provides. The functionalities that were automated were sign-in, sign-out, search, adding an item to cart, adding an item to a list, navigation to the returns page and navigation to the promotions page.

This project report is organized as follow: Section 2 focuses on the literature review of the automation regression testing. Section 3 is focused on the tools and techniques and research methodology for the project. Section 4 contains results of execution of the test suites for this project. 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 project is provided.

2. LITERATURE REVIEW

In the past there were many researches carried out on regression test case prioritisation with different techniques implemented. Regression testing is performed to make sure that any changes to an application do not affect the existing functions. Test case prioritisation is an important technique to increase efficiency of regression tests and reduce time and costs. These existing prioritisation techniques are described in multiple different studies.

One study presented a prioritisation technique that filters tests through Information Retrieval and compares the differences between two program versions, implemented through their tool REPiR [1]. This resulted in more computational efficient testing that performs better than existing prioritisation techniques.

Another study used a prioritisation technique through a regression test selection tool called TestRank, which works through dynamic and natural language analyses of a code base and its test suite and outputting a list tests ranked by relevance to changes [2]. The results showed that the technique accelerated retesting during development while maintaining high fault detection.

Similarly, another study proposed using a combination of both prioritisation techniques and regression test selection [3]. Through an algorithm, the test cases with high priority are prioritised, and then through another algorithm, the tests are selected for regression. The results showed that the technique could reduce the number of test cases and therefore the cost and resources for performing regression testing.

A different prioritisation technique that is proposed showed that instead of relying on software code information as the majority of regression testing techniques do, suggested incorporating a requirements-based clustering approach, where textual similarities in requirements a grouped together and prioritised [4]. Results showed that using information about requirements during the test case prioritization process could be beneficial.

Another requirements-based prioritisation technique that was recommended was the Priority of Requirements for Test (PORT) [5]. PORT analyses requirements volatility, customer-assigned priority, implementation complexity and fault proneness of the requirements, and uses these factors to prioritise the test cases. Results showed improvement in rate of detection of severe faults, and that customer priority was a key prioritisation factor contributing to improved fault detection rates.
Another study proposed a prioritisation technique that orders and prioritises the test cases based on

the total code coverage through information gathered about the previous execution of test cases [6]. The results of this approach showed improvement in the rate of fault detection of test suites.

A study is conducted to focus on prioritization technique that orders the test cases in a test suite such that it minimises the lines of code needed to be re-executed [7]. This results in faster code coverage, leading to early detection of faults.

A proposed prioritisation approach was using a risk-based test case prioritisation technique that focuses on new test cases [8]. The priority is decided according to test prioritisation calculated through requirement analysis and using these values to evaluate relevant test cases and determine their priority. Results show that the technique is effective in prioritising severe faults.

A different prioritisation technique was proposed that uses a static black-box test case prioritisation technique that represents test cases using the linguistic data of the test cases [9]. The linguistic data such as name, identifier and comments are analysed through a test analysis algorithm called topic modelling to approximate the functionality of a test case and give priority to test cases that test different functions. The results were that the technique was an effective way to statically prioritize test cases, while being lightweight.

Due to limitations in documentation and resources, the prioritisation techniques suggested in these studies cannot be implemented in this project. To apply these techniques, the documentation for the requirements, version change and defects of PB tech's web application are needed, which were not available for access.

3. PROJECT EXECUTION

3.1. Project Planning

In this section we will discuss about tools and proposed architecture of automation framework for this project.

3.1.1. Tools and Techniques

The regression testing automation was performed using Selenium WebDriver. Selenium is an open source tool and is considered the most popular for testing web applications [10].The primary reason to choose Selenium was due to the fact that it is an open-source tool, which means there are no additional costs required.

An add-on to Selenium WebDriver was also used called TestNG. TestNG is an automation testing framework that helps with execution of the Selenium test scripts and generates a test execution report. It provides features such as TestNG listeners for tracking the test scripts that are passed, failed and skipped, and assertions to compare expected results with actual results during script execution [11]. TestNG also provides generating of test execution reports, which was necessary for this project, as one of the drawbacks of Selenium WebDriver is that it does not natively have a test reporting feature. Parallel execution of tests can also be accomplished with TestNG to further speed up test execution.

The Page Object Model (POM) pattern was used for writing the automation test scripts. The POM model allows the mapping of pages of the web application to a page object. This pattern helps to enhance the tests, making them more maintainable, reducing code duplication, building a layer of abstraction, and hiding the inner implementation from tests [12]. This is especially helpful for regression testing, as the page objects are easily accessible and can be reused for different functionality tests.

3.1.2. Methodology for Project

The methodology chosen for the project was the Agile SCRUM methodology. Scrum was chosen over more traditional methodologies as it can lead to productivity benefits in projects, an increase in customer satisfaction, product and process quality, team motivation and cost reduction [13].

Another reason that Scrum was chosen was, Scrum methodology is ideal for rapidly changing and accumulating requirements and is fast, quick and can adapt to changes easily [14]. This was suitable for PB Tech's web application as there were many changes to the requirements occurring frequently due to customer demand.

3.1.3. Test Environment

The test environment for the project is given below. System: Intel i5-8265U 1.6GHz
NVIDIA GeForce MX130 2GB VRAM
12GB DDR4 memory 1000GB HDD
Test Data: a valid username and password for login is required Operating system: Windows 10 (x64)
Browser: Google Chrome 77.0.3865.90 (x64)

Dependencies: the test scripts written must ensure that each test is independent from each other so that the true status of the test can be found. Also, to ensure the possibility for running parallel tests in the future, the tests must be independent.

3.2. Proposed Architecture of Automation Framework

The automation framework proposed for this project is shown in Figure 1. Figure 1 shows that the architecture implemented for the PB Tech project was the Page Object Model (POM) pattern. The POM pattern consists of page object classes that allow easier maintenance and increased reusability of code. Each page of the PB Tech web application was given a separate page object. These page objects were then reusable for test cases with different functionalities.

Selenium WebDriver was used to send commands through the web browser to the web application under test. All test cases were run through the Test Suite, which required the use of the TestNG framework to be implemented.

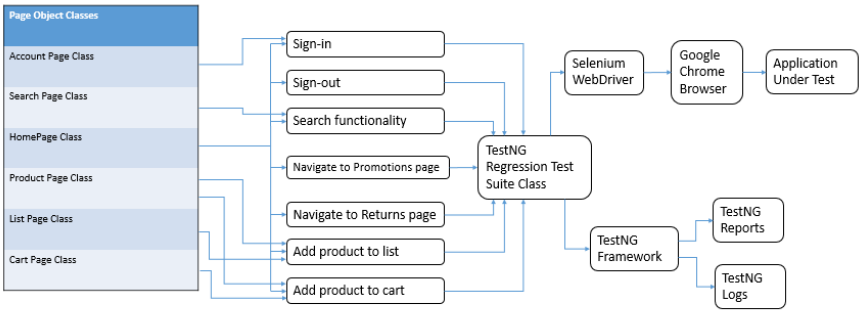

Figure 1. Proposed Regression Test Suite Automation Framework

3.3. Test Cases for Project

The test cases that were considered for implementing into the regression test suite are given in Table 1. These test cases were selected through human evaluation and represent the full regression test suite for PB Tech's website. Table 1 shows the test cases selected were: the functions of login, homepage hero image, department header menu, brand header menu, promotions page, clearance page, search bar, store finder, product returns page, careers page, product wish list, shopping cart and stock availability checker.

Table 1. Test Cases for Project

Test Case ID	Test Case Description
TC01	User can sign-in to PB Tech website
TC02	User can click on Hero Image on homepage
TC03	User can find product by department
TC04	User can find product by brand
TC05	User can click and view products with promotions
TC06	User can click and view products on clearance
TC07	User can search for specific term in the search bar and sort by "Most Popular"
TC08	User can view store finder page
TC09	User can view returns policy page
TC10	User can view careers page
TC11	User can add a product to a list
TC12	User can add products to the shopping cart
TC13	User can check stock availability of a product
TC14	User can sign-out from website

3.4. Test Case Prioritisation and Selection

The prioritisation technique implemented was a combination of human evaluation-based test case prioritisation [15], and a newly proposed prioritisation method based on the results of a study that analysed the top 100 retailer websites to assess which online features are present in the websites with the highest earnings [16]. The features of PB Tech's website were compared and the features that could be mapped to the features used in the study were selected for prioritisation. The mapped product related online features were registration/sign-in, wish list, and search; the distribution related online features were the shopping cart and return policy; and the promotion related features were promotions/special offers. For the features that were not selected for the prioritised regression suite, only human-based test case prioritisation is implemented, where test cases are prioritised according to human evaluation.

Table 2. Prioritised Test Cases

Test Case ID	Test Case Priority
TC01	High
TC02	Low
TC03	Medium
TC04	Medium
TC05	High
TC06	Medium
TC07	High
TC08	Medium
TC09	High
TC10	Medium
TC11	High
TC12	High
TC13	Medium
TC14	High

The test cases were then assigned a priority according to the prioritisation technique, shown in Table 2. Table 2 shows the test cases prioritised according to the feature's earnings and they were assigned as high priority. The test cases such as product navigation by brand and department, clearance products page, store finder, careers page and stock availability check were assigned as medium priority while the homepage hero image was assigned as low priority.

After the test cases were prioritised accordingly, test case selection was performed. This was done by selecting the test cases that were given a high priority. These were compiled into the prioritised regression test suite.

3.5. Automation Test Script Design

Automation test scripts were created for each of the test cases selected for this project. An example of this is shown in Figures 2 and 3. Figure 2 displays the automation script of test case TC01, created according to the TestNG framework. This class calls an instance object of the Account page class, found in Figure 3. The automation scripts were structured using the POM pattern, which enables the page object classes to be reusable for other tests in the future.

```java
package pbtech.pages;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.CacheLookup;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;
import org.openqa.selenium.support.PageFactory;

public class AccountPage {
    WebDriver driver;

    @FindBy(how = How.XPATH, using = "//input[@id='try_login']")
    @CacheLookup
    WebElement emailField;

    @FindBy(how = How.XPATH, using = "//input[@id='try_pass']")
    @CacheLookup
    WebElement pwField;

    @FindBy(how = How.XPATH, using = "//button[contains(text(), 'Sign In')]")
    @CacheLookup
    WebElement loginBtn;

    @FindBy(how = How.XPATH, using = "//div[@class='headButtons']/div[@class='header_button_float'][2]")
    @CacheLookup
    WebElement userTab;

    @FindBy(how = How.XPATH, using = "//a[@href='logout_pdo.php']")
    @CacheLookup
    WebElement logoutBtn;


    public AccountPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public void login(String email, String pw) {
        emailField.sendKeys(email);
        pwField.sendKeys(pw);
        loginBtn.click();
    }

    public void logout() throws InterruptedException {
        Actions action = new Actions(driver);
        action.moveToElement(userTab).build().perform();
        Utilities.WAIT_TWO_SEC();
        logoutBtn.click();
    }
}
```

Figure 2. Account Page Object Class

```java
package pbtech.test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.Assert;
import pbtech.pages.HomePage;
import pbtech.pages.AccountPage;
import pbtech.pages.Utilities;

public class TestSignIn {
    WebDriver driver;
    HomePage homepage;
    AccountPage account;

    String email = "agitest708@gmail.com";
    String pw = "AGItestacc";
    String name = "Billy";
    String logoutSuccess = "Create Account";

    @BeforeSuite
    public void setup() {
        System.setProperty("webdriver.chrome.driver", "C:\\Temp\\chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("https://www.pbtech.co.nz/");
        driver.manage().window().maximize();
        homepage = new HomePage(driver);
    }

    @Test
    public void loginTest() throws InterruptedException {
        //Utilities.WAIT_ONE_SEC();
        account = homepage.clickSignIn();
        Utilities.WAIT_ONE_SEC();
        String actual1 = driver.findElement(By.xpath("//div[@id='newRegForm']/h2")).getText();
        String expected1 = "SIGN IN DETAILS";
        Assert.assertEquals(actual1, expected1);

        account.login(email, pw);
        Utilities.WAIT_ONE_SEC();
        String actual2 = driver.findElement(By.xpath("//a[@href='my-account']")).getText();
        String expected2 = name;
        Assert.assertEquals(actual2, expected2);
    }

    @AfterTest
    public void logout() throws InterruptedException{
        Utilities.WAIT_ONE_SEC();
        account.logout();
        Utilities.WAIT_ONE_SEC();
    }

    @AfterSuite
    public void tearDown() {
        driver.quit();
        driver = null;
    }
}
```

Figure 3. Sign-In Test Class

Two different test suites were created in the process: the full regression test suite with all test cases, and the prioritised regression test suite with only the prioritised test cases. The test suites were then executed and TestNG reports were generated.

4. RESULTS

The results of execution of the test suites results in the generation of TestNG reports. The reports that were analysed during this project were the emailable report, test times report and the method execution chronological order report. The emailable report contains the total time taken to run the suite, the individual times taken for each test and the number of passed, failed and skipped tests. The test times report contains the total running time and the individual test running time in decreasing order. The methods in chronological order report shows in detail which method of each test is executed and at what times they are started. It also shows the start and end time of the test including the driver setup and closure, which the other reports do not show.

The emailable report generated for the prioritised regression test suite is shown in Figure 4. Figure 4 shows that all test scripts passed, with no failed or skipped tests. The total time taken to execute the test suite was 73.6 seconds and the individual times taken for each test is also shown.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
TCPRegressionSuite						
Test	7	0	0	73,620		
allSuites						

Class	Method	Start	Time (ms)
TCPRegressionSuite			
Test — passed			
pbtech.test.TestCart	cartTest	1569075150531	9066
pbtech.test.TestList	listTest	1569075137867	12663
pbtech.test.TestPromotions	promotionsTest	1569075120309	3384
pbtech.test.TestReturns	returnsTest	1569075134556	3310
pbtech.test.TestSearch	searchTest	1569075123694	10861
pbtech.test.TestSignIn	loginTest	1569075115390	4911
pbtech.test.TestSignOut	logoutTest	1569075159598	5437
allSuites			

Figure 4. Prioritised Regression Test Suite Emailable Report

The result of the execution of the full regression test suite is given in Figure 5. Figure 5 shows that all tests passed and the time taken to execute was 122.5 seconds. By prioritising the test cases, there was a reduction of 48.8 seconds or 40% in total execution time.

The test time report, in decreasing order, of each test of the prioritised regression test suite are shown in Figure 6. Figure 6 shows that the list functionality test took the longest at 12.7 seconds and the returns page took the shortest time at 3.3 seconds.

The times taken for the tests from the full regression test suite are shown in Figure 7. Figure 7 shows that the list functionality test also took the longest at 12.7 seconds and the find by brand test took the shortest time at 3.4 seconds. However, the total running time is given in minutes, which is imprecise compared to the times shown in the emailable reports.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
CompleteRegressionSuite						
Test	14	0	0	122,481		
allSuites						

Class	Method	Start	Time (ms)
CompleteRegressionSuite			
Test — passed			
pbtech.test.TestBrand	brandTest	1569080862098	3391
pbtech.test.TestCareers	careersTest	1569080894674	3843
pbtech.test.TestCart	cartTest	1569080911232	9470
pbtech.test.TestClearance	clearanceTest	1569080868980	5652
pbtech.test.TestDepartment	departTest	1569080856224	5874
pbtech.test.TestHomepage	homepageTest	1569080850012	6211
pbtech.test.TestList	listTest	1569080898517	12714
pbtech.test.TestPromotions	promotionsTest	1569080865490	3490
pbtech.test.TestReturns	returnsTest	1569080891265	3408
pbtech.test.TestSearch	searchTest	1569080874633	11700
pbtech.test.TestSignIn	loginTest	1569080841973	8037
pbtech.test.TestSignOut	logoutTest	1569080927347	5445
pbtech.test.TestStock	stockCheckTest	1569080920703	6643
pbtech.test.TestStoreFinder	storeFinderTest	1569080886334	4930
allSuites			

Figure 5. Full Regression Test Suite Emailable Report

Times for TCPRegressionSuite

Total running time: 49 seconds

Number	Method	Class	Time (ms)
0	listTest	pbtech.test.TestList	12,663
1	searchTest	pbtech.test.TestSearch	10,861
2	cartTest	pbtech.test.TestCart	9,066
3	logoutTest	pbtech.test.TestSignOut	5,437
4	loginTest	pbtech.test.TestSignIn	4,911
5	promotionsTest	pbtech.test.TestPromotions	3,384
6	returnsTest	pbtech.test.TestReturns	3,310

Figure 6. Times for Prioritised Regression Test Suite

Times for CompleteRegressionSuite

Total running time: 1 minutes

Number	Method	Class	Time (ms)
0	listTest	pbtech.test.TestList	12,714
1	searchTest	pbtech.test.TestSearch	11,700
2	cartTest	pbtech.test.TestCart	9,470
3	loginTest	pbtech.test.TestSignIn	8,037
4	stockCheckTest	pbtech.test.TestStock	6,643
5	homepageTest	pbtech.test.TestHomepage	6,211
6	departTest	pbtech.test.TestDepartment	5,874
7	clearanceTest	pbtech.test.TestClearance	5,652
8	logoutTest	pbtech.test.TestSignOut	5,445
9	storeFinderTest	pbtech.test.TestStoreFinder	4,930
10	careersTest	pbtech.test.TestCareers	3,843
11	promotionsTest	pbtech.test.TestPromotions	3,490
12	returnsTest	pbtech.test.TestReturns	3,408
13	brandTest	pbtech.test.TestBrand	3,391

Figure 7. Times for Full Regression Test Suite

The execution times of each method of the prioritised regression test suite, in chronological order, are given in Figure 8. Figure 8 shows that the initial driver setup methods of the tests were all executed at the start and took 84.4 seconds to complete, the driver closure methods took 3.8 seconds and the total method running time was 161.8 seconds.

The times of each method of the full regression test suite is given in Figure 9. Figure 9 shows that the driver setup methods took 168 seconds to complete, the driver closure methods took 8.1 seconds and the total method running time 298.6 seconds.

The decrease in time taken for both the setup and closure methods is around 50%. This indicates that there is a linear increase proportional to the increase in tests executed. There also is a decrease in the total method execution time by 46%.

Methods in chronological order	
pbtech.test.TestCart	
setup	0 ms
pbtech.test.TestList	
setup	12979 ms
pbtech.test.TestPromotions	
setup	25087 ms
pbtech.test.TestReturns	
setup	36981 ms
pbtech.test.TestSearch	
setup	48789 ms
pbtech.test.TestSignIn	
setup	60724 ms
pbtech.test.TestSignOut	
setup	72549 ms
pbtech.test.TestCart	
login	84373 ms
pbtech.test.TestList	
login	86337 ms
pbtech.test.TestSignOut	
login	89092 ms
pbtech.test.TestSignIn	
loginTest	94162 ms
pbtech.test.TestPromotions	
promotionsTest	99081 ms
pbtech.test.TestSearch	
searchTest	102466 ms
pbtech.test.TestReturns	
returnsTest	113328 ms
pbtech.test.TestList	
listTest	116639 ms
pbtech.test.TestCart	
cartTest	129303 ms
pbtech.test.TestSignOut	
logoutTest	138370 ms
pbtech.test.TestCart	
logout	143808 ms
pbtech.test.TestList	
logout	148211 ms
pbtech.test.TestSignIn	
logout	152491 ms
pbtech.test.TestCart	
tearDown	158020 ms
pbtech.test.TestList	
tearDown	158696 ms
pbtech.test.TestPromotions	
tearDown	159336 ms
pbtech.test.TestReturns	
tearDown	159924 ms
pbtech.test.TestSearch	
tearDown	160536 ms
pbtech.test.TestSignIn	
tearDown	161147 ms
pbtech.test.TestSignOut	
tearDown	161774 ms

Figure 8. Prioritised Regression Test Suite Methods in Chronological Order

Methods in chronological order	
pbtech.test.TestBrand setup	0 ms
pbtech.test.TestCareers setup	13415 ms
pbtech.test.TestCart setup	25409 ms
pbtech.test.TestClearance setup	37413 ms
pbtech.test.TestDepartment setup	49639 ms
pbtech.test.TestHomepage setup	61243 ms
pbtech.test.TestList setup	73179 ms
pbtech.test.TestPromotions setup	84956 ms
pbtech.test.TestReturns setup	96662 ms
pbtech.test.TestSearch setup	108938 ms
pbtech.test.TestSignIn setup	121331 ms
pbtech.test.TestSignOut setup	134247 ms
pbtech.test.TestStock setup	146635 ms
pbtech.test.TestStoreFinder setup	158258 ms
pbtech.test.TestCart login	167978 ms
pbtech.test.TestList login	170947 ms
pbtech.test.TestSignOut login	179519 ms
pbtech.test.TestSignIn loginTest	185141 ms
pbtech.test.TestHomepage homepageTest	193180 ms
pbtech.test.TestDepartment departTest	199392 ms
pbtech.test.TestBrand brandTest	205266 ms
pbtech.test.TestPromotions promotionsTest	208658 ms
pbtech.test.TestClearance clearanceTest	212148 ms
pbtech.test.TestSearch searchTest	217801 ms
pbtech.test.TestStoreFinder storeFinderTest	229502 ms
pbtech.test.TestReturns returnsTest	234433 ms
pbtech.test.TestCareers careersTest	237842 ms
pbtech.test.TestList listTest	241685 ms
pbtech.test.TestCart cartTest	254400 ms
pbtech.test.TestStock stockCheckTest	263871 ms
pbtech.test.TestSignOut logoutTest	270515 ms
pbtech.test.TestCart logout	275960 ms
pbtech.test.TestList logout	280383 ms
pbtech.test.TestSignIn logout	285083 ms
pbtech.test.TestBrand tearDown	290490 ms
pbtech.test.TestCareers tearDown	291088 ms
pbtech.test.TestCart tearDown	291698 ms
pbtech.test.TestClearance tearDown	292398 ms
pbtech.test.TestDepartment tearDown	293008 ms
pbtech.test.TestHomepage tearDown	293657 ms
pbtech.test.TestList tearDown	294262 ms
pbtech.test.TestPromotions tearDown	294878 ms
pbtech.test.TestReturns tearDown	295484 ms
pbtech.test.TestSearch tearDown	296118 ms
pbtech.test.TestSignIn tearDown	296778 ms
pbtech.test.TestSignOut tearDown	297392 ms
pbtech.test.TestStock tearDown	298017 ms
pbtech.test.TestStoreFinder tearDown	298626 ms

Figure 9. Full Regression Test Suite Methods in Chronological Order

5. DISCUSSION

The results show that the compared to the full regression test suite, the time taken to run the prioritised regression test suite was significantly reduced. As the full regression test suite continues to grow from new features, the time saved by the proposed technique would also continue to increase.

The results show that the execution of the tests in both the full regression test suite and the prioritised regression test suite were all passed. The execution results given from a TestNG test can result in a passed, failed or skipped status. During the execution, if a test failed, then the other tests that are dependent on the same test would have been skipped, which would have affected the results and made the time measurements inaccurate. However, this was prevented by making all the tests independent, and the result of this project was that all tests passed with no failed or skipped tests.

A comparison with previous studies is difficult as the performance indicators used in those studies are based on comparing fault detection rate with previous versions. This is the case for many of the studies, such as in the study of Saha et al., where they used Average Percentage Faults Detected as a metric for prioritization effectiveness [1]. However, in this project, the sole performance indicator was the duration of the test suites. Therefore, it is not possible to give a direct comparison with the prioritisation techniques found in the previous studies.

As an alternative to the many different test case prioritisation techniques that exist, the technique proposed in this report was able to reduce the time taken without the need for fault rate information or requirements analysis. This technique could be helpful as an alternative method to use for projects that are missing the required documentation in order to compare the fault detection rate or perform requirements based test case prioritisation techniques.

6. RECOMMENDATIONS

During this project, one of the most prominent problems encountered was the lack of documentation of PB Tech's web application requirements, defects and complete regression suite. Due to this, there was no feasible method to implement other test case prioritisation techniques to compare to. This was due to having no access to PB Tech's project and systems. The only solution to this issue to is by gaining authorisation to access their systems and documentation.

Another potential issue that was observed was the automation tests were only executing 7 test cases but took 73.6 seconds to complete. This could be due to hardware limitations of the test environment that was implemented. A potential solution is to upgrade the hardware to improve the performance of the tests. Another solution could be to execute the tests in parallel, which could save more time.

A factor that needs consideration is, the time taken to perform a human evaluation-based test case prioritisation technique may be significantly slower compared to other prioritisation techniques and could offset the overall time saved. To measure the amount of time lost due to this would require further analysis. This can possibly be solved through fully automating the prioritisation process with statistical data instead of human evaluation. However, this may become costly to implement. Further analysis into an alternative solution is needed.

7. CONCLUSION

This project demonstrated the effectiveness of the proposed test case prioritisation technique by significantly reducing the time taken to execute the prioritised regression suite compared to full regression testing. This also confirms that human evaluation-based and statistics based prioritisation is a viable technique in reducing testing times. This project also indicated the reusability of automation scripts which allows easier and faster repeated testing. These findings are significant as they show how automation and test case prioritisation can greatly improve the regression testing process and alleviate repetitive tests

REFERENCES

[1] Saha, R. K., Zhang, L., Khurshid, S., & Perry, D. E. (2015). An information retrieval approach for regression test prioritization based on program changes. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 1, 268-279.

[2] Cibulski, H., & Yehudai, A. (2011). Regression test selection techniques for test-driven development. In 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 115-124.

[3] Malhotra, R., Kaur, A., & Singh, Y. (2010). A Regression Test Selection and Prioritization. Journal of Information Processing Systems, 6(2), 235-252.

[4] Arafeen, M. J., & Do, H. (2013). Test case prioritization using requirements-based clustering. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 312-321.

[5] Srikanth, H., Williams, L., & Osborne, J. (2005). System test case prioritization of new and regression test cases. In 2005 International Symposium on Empirical Software Engineering, 10.

[6] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. IEEE Transactions on software engineering, 27(10), 929-948.

[7] Badhera, U., Purohit, G. N., & Biswas, D. (2012). Test case prioritization algorithm based upon modified code coverage in regression testing. International Journal of Software Engineering & Applications, 3(6), 29.

[8] Yoon, M., Lee, E., Song, M., & Choi, B. (2012). A test case prioritization through correlation of requirement and risk. Journal of Software Engineering and Applications, 5(10), 823.

[9] Thomas, S. W., Hemmati, H., Hassan, A. E., & Blostein, D. (2014). Static test case prioritization using topic models. Empirical Software Engineering, 19(1), 182-212.

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

[11] 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.

[12] Vila, E., Novakova, G., & Todorova, D. (2017). Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats. International Conference on Advances in Image Processing, 144-150.

[13] Cardozo, E., Neto, J., Barza, A., França, A. & da Silva, F. (2010). SCRUM and Productivity in Software Projects: A Systematic Literature Review. EASE.

[14] Mahalakshmi, M., & Sundararajan, M. (2013). Traditional SDLC Vs Scrum Methodology – A Comparative Study. International Journal of Emerging Technology and Advanced Engineering, 3(6), 192-196.

[15] Saklani, N. K., Singh, P. (2017). Review of Prioritization Techniques in Regression Testing. International Journal of Computer Science and Mobile Computing, 6(4), 216-221.

[16] Girard, T., Anitsal, M. M., & Anitsal, I. (2008). Online features of the top 100 US retailers' web sites. The International Journal of the Academic Business World, 2(1), 9-1.