AUTOMATED REGRESSION TESTS AND AUTO-MATED TEST OPTIMISATION FOR GETRV

Neil Kevin Patalita Arcolas[1] and Shahid Ali[2]

[1]Department of Information Technology, AGI Institute, Auckland, New Zealand
[2] Department of Information Technology, AGI Institute,
Auckland, New Zealand

ABSTRACT

Regression testing is a type of testing that is performed to validate that new changes pushed to the system does not have any adverse effect to the existing features. Automated regression testing greatly reduces the time spent by testers to perform these repetitive and mundane tests and allows them to work on more critical tests. The first problem addressed in this project is to add two automated regression scripts to increase test coverage of the existing test automation framework. The second problem is to optimise the automated regression test run to reduce the test run times. Additionally, to improve the automated test run times, redundant expressions were removed and handled in the outermost loop of the automated test run. The project resulted in the addition of two automated test scripts for the automated test run and a significant test run time reduction of at least 60%.

KEYWORDS

Automated regression, Agile scrum, Automated test run

1. INTRODUCTION

Information Technology enabled services and web applications are ubiquitous nowadays. The range and areas of application for web-enabled services and applications are ever-increasing. It is commonplace to find web-based applications in short-term home rentals, an example being AirBnB, hotel booking like Trivago, restaurant rating like Zomato and so. This project was carried out for GetRV company, the largest provider of Recreational Vehicles (RV's) both for sales and for rent in Australia and New Zealand. The company currently has a web application that serves as a platform for their RV providers to create and manage their products, product contents, pricing, scheduling, and booking.

GetRV's web application is subject to changes throughout its development and post-development phases. Reasons for these changes range from changes in requirements, updates in the underlying technologies used by these applications, and an addition of new features. Regression testing has become a necessary testing practice to make sure the changes implemented are not affecting the parts of the software application that are not part of the changes [1]. Since regression testing is essentially doing the same tests multiple times, it can be time consuming and mundane, albeit necessary, for the testers that are performing them. This is especially true with front end testing where testers need to interact with the front-end user interface and do these numerous times. To address this, GetRV's Quality Assurance (QA) Team has created a scenario-based regression test automation framework that is run whenever there are changes and updates to their web application.

The test automation framework allows members of the QA Team to conveniently define new test scenarios in Gherkin statements using SpecFlow. Creation of new automation test scripts for new test scenarios come with minimal to no changes to the underlying test automation framework. Additionally, it also allows the testers to structure their tests into features composed of related test scenarios for easy maintenance. The integrated NUnit NuGet will allow the testers to see the structure of the tests in a Test Explorer View Window. Additionally, NUnit is also used to run individual tests, multiple tests, or all of the regression test scenarios from their own workstation and see the returned test results in the Test Explorer View Window. The regression test automation framework is built using C# Language on a .Net Framework and using Microsoft Visual Studio 2017 as an Integrated Development Environment (IDE). The automated tests were performed on GetRV's web application in Google Chrome with the use of Coypu, a wrapper for Selenium WebDriver. The mentioned set up mimics what a manual tester would do when performing their manual testing on the user interface. Finally, the existing framework is an implementation of a page object pattern test automation framework which models the web pages as objects in the program. These objects contain the web elements that are present in the page and methods are created within the objects to perform the actions that will be done on that page.

Moreover, as the number of features of the web application increases there was a need for the following:(1) grow the number of regression tests to accommodate those changes; and (2) optimise the automated regression test run to reduce test run times. This project aims to contribute to the existing regression test scenarios by adding two new test scenarios. The first scenario is the "Update Content Details", which is a feature under the web application's Content module. This module allows users to create and manage product contents and media. These are then picked up and displayed in the Products module. The "Update Content Details" test scenario was used to test if the current web application configuration allows users to update the details of an existing content. The second scenario is the "Update Rental Details", which is a feature under the web application's Booking module. This module allows users to create and manage existing booking or rentals for a certain product. The "Update Rental Details" test scenario will test if the current web application configuration allows users to update existing rental details. Both of which are features that are already available in GetRV's web application but is currently not covered by the existing regression test scenarios. Consequentially, adding test scenarios will increase the run time of the whole test run. To address this, this project has created one test run optimisation that has reduced the test run time of each test scenario and consequently the whole test run. In addition, this project has also retrofitted existing test scenarios to accommodate the test run optimisation. Due to the limited time frame of the project, only details that are marked as mandatory in the web application's user interface have been updated for the mentioned scenarios. Additionally, only two test scenarios have been retrofitted for the test run optimisation. Those were the "Updating Existing Content's Details" and "Updating Existing Rental's Details" scenarios which were previously created in this project.

2. LITERATURE REVIEW

Over the years, many studies and publications have been reported in literature on the topic of regression testing [2], [3] and [4]. In the following, only some of the approaches discussed in the aforementioned papers in regression testing domain were surveyed. The survey is followed by a discussion of those approaches that were relevant to this project.

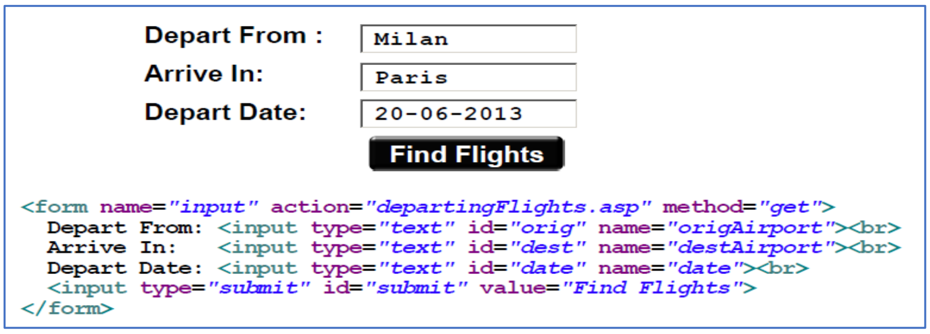
A scenario-based functional regression testing for any integrated software application was proposed [5]. This approach to regression testing closely resembles the existing approach currently implemented by GetRV. While other techniques presented in the paper are code-based using program slicing, program dependence graphs, and data flow and control flow analysis, the proposed approach on regression testing focuses on the overall functional correctness of the whole integrated software application. This type of approach will validate that all integrated components of the software system (both internal and external), will together support the intended business functions. In scenario-based testing, the test steps are structured to mimic what a business user would do in the application. To perform this type of regression testing, test scenarios are defined in thin-threads and conditions which are semi-formal representations of the defined requirements. A thin-thread is a minimum user scenario that describes what the end user will be doing in the system. Conditions are factors affecting the execution of the thin-thread's functionality. A thin-thread will serve as a basic translation of the product requirements to functional regression test scenarios having defined inputs, actions, and outputs. Additionally, similar test scenarios can be grouped together. Moreover, types of regression test selection strategy that can be implemented with scenario-based testing have been defined in the paper. One strategy that closely resembles the current approach of GetRV is "Random Testing", in which the reliability of the regression testing is observed to be proportional to the size of the regression tests. This means that the larger the number of regression test scenarios, the more reliable it is [5]. Error! Reference source not found. highlights the main features of Scenario-based functional regression testing.

Table 1. Scenario-based Functional Regression Testing [5]

Scenario-based Functional Regression Testing
Covers over-all functional correctness
Tests all integrated components that are used in the user scenario
Thin-threads are representations of user scenarios

Test automation framework for web applications was proposed which is easy to maintain and easy to read [6]. This type of test automation framework is the page object pattern. In this approach, pages are modeled into objects and all interactions to those pages are created as a method within the page object. This makes it easy to access those methods and include them as steps in the automated test cases. In their case study, they presented how the test object pattern is used in creating and executing a simplified test case for a web application that allows flight bookings.

Figure 1 shows the page view as seen in the web application and the source of the Flights Search Page.
Figure 2 shows the page view and source of the Departing Flights Page.
Figure 3shows the web elements and the actions to be performed unto them.
Figure 4 shows a comparison of how the test cases are written without page object pattern and with page object pattern. It can be noted that using the page object pattern, the creation of web elements and actions are separated from the actual test case which makes the test case readable compared to the test case that is not implemented with a page object pattern. This way, the web elements, and page actions can be reused in other test cases without the need for creating the same code and introducing clutter [6].



```
<form name="input" action="departingFlights.asp" method="get">
  Depart From: <input type="text" id="orig" name="origAirport"><br>
  Arrive In:   <input type="text" id="dest" name="destAirport"><br>
  Depart Date: <input type="text" id="date" name="date"><br>
  <input type="submit" id="submit" value="Find Flights">
</form>
```

Figure 1. Flight Search page and source [6]

Departure		Arrival		Prices
Milan	08:30	Paris	09:45	89€
Milan	09:30	Paris	10:45	79€

```
<table id="results" border="1"><tr>
   <td colspan="2">Departure</td>
   <td colspan="2">Arrival</td>
   <td>Prices</td>
 </tr><tr>
   <td id="o1">Milan</td><td>08:30</td>
   <td id="d1">Paris</td><td>09:45</td><td>89€</td>
 </tr><tr>
   <td id="o2">Milan</td><td>09:30</td>
   <td id="d2">Paris</td><td>10:45</td><td>79€</td>
 </tr>
</table>
```

Figure 2. Departing flights page and source [6]

```
public class SearchFlightsPage {
 private final WebDriver driver;
 public SearchFlightsPage(WebDriver driver) {this.driver = driver;}
 public DepartingFlightsPage searchFlights(String orig, String dest,
                                           Date date) {
  driver.get("http://www.....com/searchFlights.asp");
  driver.findElement(By.id("orig")).sendKeys(orig);
  driver.findElement(By.id("dest")).sendKeys(dest);
  driver.findElement(By.id("date")).sendKeys(date.toString());
  driver.findElement(By.id("submit")).submit();
  return new DepartingFlightsPage(driver);
 }
}
```

Figure 3. Web Elements and Page Actions [6]

```
public void testEuropeanFlight() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'searchFlights.asp' page
 driver.get("http://www.....com/searchFlights.asp");
 driver.findElement(By.id("orig")).sendKeys("Milan");
 driver.findElement(By.id("dest")).sendKeys("Paris");
 driver.findElement(By.id("date")).sendKeys("20-06-2013");
 driver.findElement(By.id("submit")).submit();
 // we are in the 'departingFlights.asp' page
 for each flight X returned, X = [1..n] {      //(o1,d1)...(on,dn)
  assertEquals("Milan", driver.findElement(By.id("oX")).getText());
  assertEquals("Paris", driver.findElement(By.id("dX")).getText());
 }
 driver.close();
}
```

```
public void testEuropeanFlight() {
 WebDriver driver = new FirefoxDriver();
 // we start from the 'searchFlights.asp' page
 SearchFlightsPage SFP = new SearchFlightsPage(driver);
 DepartingFlightsPage DFP = SFP.searchFlights("Milan","Paris",
                        new Date(20,06,2013));
 // we are in the 'departingFlights.asp' page
 for each flight X returned, X = [1..n] {
  assertEquals("Milan", DFP.getOrigFlight(X));
  assertEquals("Paris", DFP.getDestFlight(X));
 }
 driver.close();
}
```

Figure 4. Comparison of Search flights test without page object pattern (above) and with page object pattern (below) [6]

A code optimisation technique [7] that is applied in this project to reduce the runtimes of the test scenarios. The proposed technique [7] is used to move redundant expressions to the entry of the outermost loop where it is invariant or never changing. This avoids the code unnecessarily redoing the calculations within the inner loops or path of the program. An experiment was performed applying their proposed code optimisation and it was found to result in 30% to 60% decrease in the run times of the code in observation and established that their proposed code optimisation does not diminish the effectiveness of the code. (Morel & Renvoise, 1979).

The chosen methodology for this project is the Agile Scrum methodology. GetRV is already following this type of methodology. The implementation and benefits of Agile Scrum methodology have been discussed in literature by Schwaber. Agile Scrum is described as a type of iterative methodology which assumes that the software development process is an unpredictable and complicated process that can be approximately described as an overall progression. In contrast to the waterfall process that is linear in nature and does not define how to handle any unexpected output from its adjacent processes, Scrum methodology is flexible to unpredictable outcomes of its processes and employs continuous review and feedback of both the process and the artefacts involved. The main phases of the Scrum methodology are shown in Figure 5. Since review and feedback is necessary with this type of process, testing is done in parallel with the development to shorten the feedback loop when there are issues that need to be addressed. This is especially true in developing modern web applications where businesses usually have more features to add or tweak when they have already seen the application in use [8].

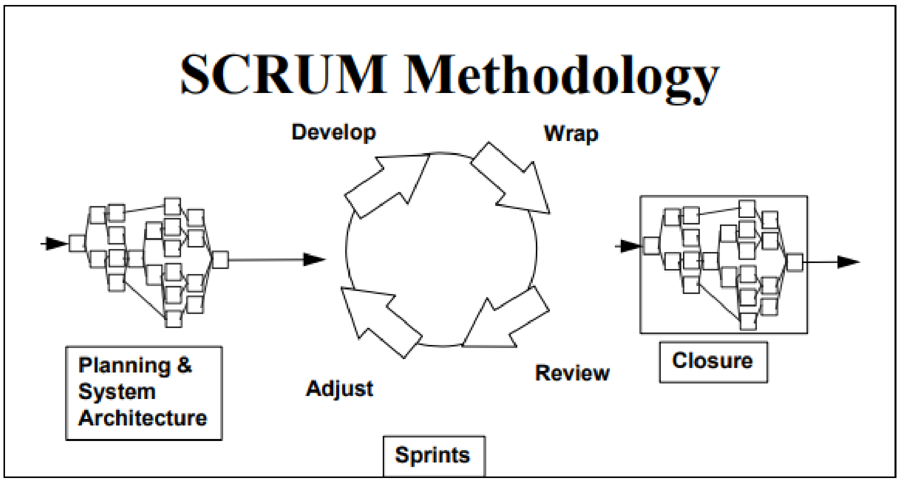


Figure 5. SCRUM Methodology [8]

4. PROJECT EXECUTION

This section describes the actual execution of the project and the application of the knowledge from the reviewed literature. This project leveraged the existing processes and tools that are currently used by GetRV. The existing STLC (Software Testing Life Cycle) used in GetRV is one that is integrated into their existing Scrum methodology and are done in *Grooming, Planning, Sprint, and Release* phases. The five-week project was divided into three sprints: Sprint 0, Sprint 1, and Sprint 2. Sprint 0 was only for one week because it was reserved for the familiarisation of the organisation, its processes and best practice, and the tools that they are using, the finalisation of the project proposal, and the Grooming and Planning phases of Scrum. Sprint 1 and Sprint 2 were two weeks long. Both Sprint 1 and Sprint 2 were the actual sprints unlike Sprint 0. In addition, Sprints 1 and 2 each had their own release. For the purposes of this project, the Scrum phases were structured as shown in Error! Reference source not found.

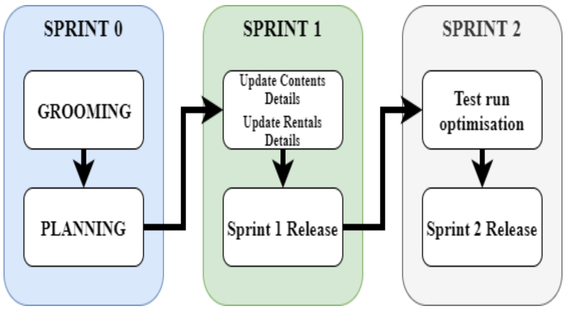


Figure 6. Project Scrum Phases

4.1. Grooming Phase

This phase covered the establishment of a communication plan, the introduction of the existing regression test automation architecture and framework, the creation of Epics that will be planned into the sprints, and introduction to additional tools. This phase was done during Sprint 0.

4.2. Planning Phase

Similarly, with Grooming, this phase was done in Sprint 0. During this phase, the Epics were planned in the remaining sprints (Sprints 1 and 2). Sprints 1 and 2 each had their own release phase because the Epic in Sprint 2 were dependent on Sprint 1. Error! Reference source not found. shows how the Epics were planned into Sprint 1 and 2 including their start and end dates, and the allocated time to complete the Epics.

4.3. Sprints Phase

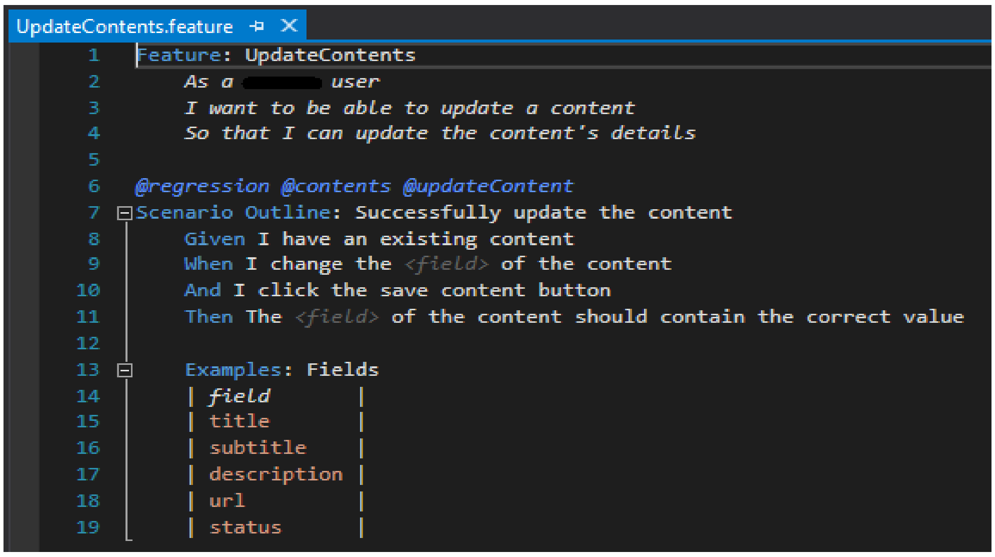
During these phases, tasks have been carried out to complete the Epics planned for each sprint. In the following subsections, the main tasks carried out in each Sprint are briefly described.

4.3.1. Sprint 1

A test scenario was created for each epic as shown in . Each of the test scenarios were then given Gherkin statements that describes the test steps and written in a SpecFlow feature file as shown in Figure for the "Update Content Details" and Figure 6 for "Update Rental Details" tests. Both scenarios were written in a Scenario Outline. This will repeat the tests depending on the number of Examples taking in the examples as inputs. These tests are then available in the test explorer and runner as shown in Figure 7.

Table 2. Epics mapped to Test Scenarios

Epics	Test Scenarios
Update Contents Details	**Successfully update the content**
Update Rentals Details	**Successfully update the rental**



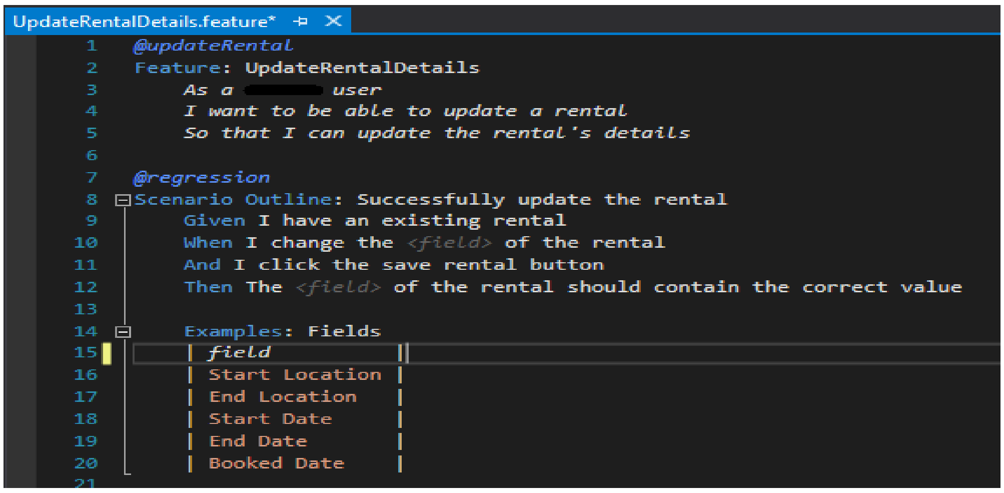
```
UpdateContents.feature

Feature: UpdateContents
    As a [redacted] user
    I want to be able to update a content
    So that I can update the content's details

@regression @contents @updateContent
Scenario Outline: Successfully update the content
    Given I have an existing content
    When I change the <field> of the content
    And I click the save content button
    Then The <field> of the content should contain the correct value

    Examples: Fields
    | field       |
    | title       |
    | subtitle    |
    | description |
    | url         |
    | status      |
```

Figure 9. Update Content Details Feature file



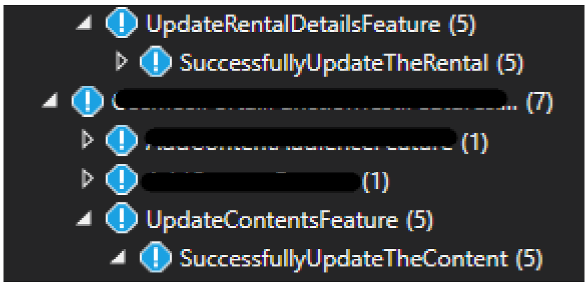
```
UpdateRentalDetails.feature*

@updateRental
Feature: UpdateRentalDetails
    As a [redacted] user
    I want to be able to update a rental
    So that I can update the rental's details

@regression
Scenario Outline: Successfully update the rental
    Given I have an existing rental
    When I change the <field> of the rental
    And I click the save rental button
    Then The <field> of the rental should contain the correct value

    Examples: Fields
    | field          |
    | Start Location |
    | End Location   |
    | Start Date     |
    | End Date       |
    | Booked Date    |
```

Figure 6. Update Rental Details Feature file

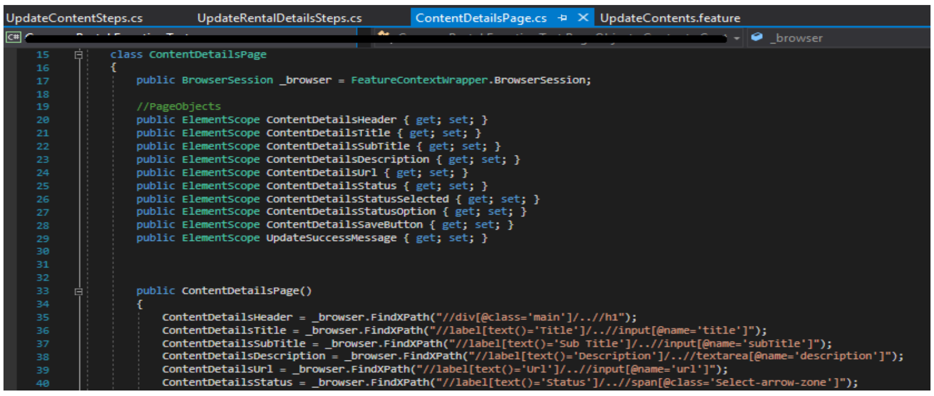


```
UpdateRentalDetailsFeature (5)
    SuccessfullyUpdateTheRental (5)
[redacted] (7)
    [redacted] (1)
    [redacted] (1)
    UpdateContentsFeature (5)
        SuccessfullyUpdateTheContent (5)
```

Figure 7. Test explorer and runner

The first Epic that was addressed is named "**Update Contents Details**". This included creating a page object class as shown in

Figure 8 and the steps definition class as shown in Figure 9. These code changes were then pushed to a working branch in GitHub and a review and pull request was sent to the industry supervisor as shown in Figure 10. While waiting for the review comments or approval for the pull request, the next Epic which is "**Update Rental Details**" was started. This also involved the creation of the page object class, step definitions class and subsequently proceeding with the code push, then a review request and pull request was sent to the industry supervisor. Review inputs from the industry supervisor were then applied to the code and further requests were made for further reviews until there were no more review comments.



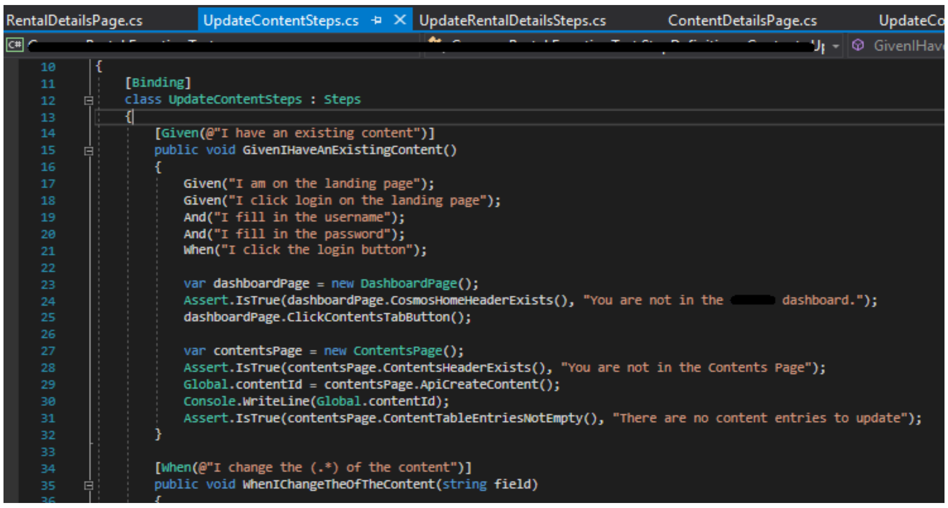
```
class ContentDetailsPage
{
    public BrowserSession _browser = FeatureContextWrapper.BrowserSession;

    //PageObjects
    public ElementScope ContentDetailsHeader { get; set; }
    public ElementScope ContentDetailsTitle { get; set; }
    public ElementScope ContentDetailsSubTitle { get; set; }
    public ElementScope ContentDetailsDescription { get; set; }
    public ElementScope ContentDetailsUrl { get; set; }
    public ElementScope ContentDetailsStatus { get; set; }
    public ElementScope ContentDetailsStatusSelected { get; set; }
    public ElementScope ContentDetailsStatusOption { get; set; }
    public ElementScope ContentDetailsSaveButton { get; set; }
    public ElementScope UpdateSuccessMessage { get; set; }



    public ContentDetailsPage()
    {
        ContentDetailsHeader = _browser.FindXPath("//div[@class='main']/..//h1");
        ContentDetailsTitle = _browser.FindXPath("//label[text()='Title']/..//input[@name='title']");
        ContentDetailsSubTitle = _browser.FindXPath("//label[text()='Sub Title']/..//input[@name='subTitle']");
        ContentDetailsDescription = _browser.FindXPath("//label[text()='Description']/..//textarea[@name='description']");
        ContentDetailsUrl = _browser.FindXPath("//label[text()='Url']/..//input[@name='url']");
        ContentDetailsStatus = _browser.FindXPath("//label[text()='Status']/..//span[@class='Select-arrow-zone']");
```

Figure 8. Content Details Page object class



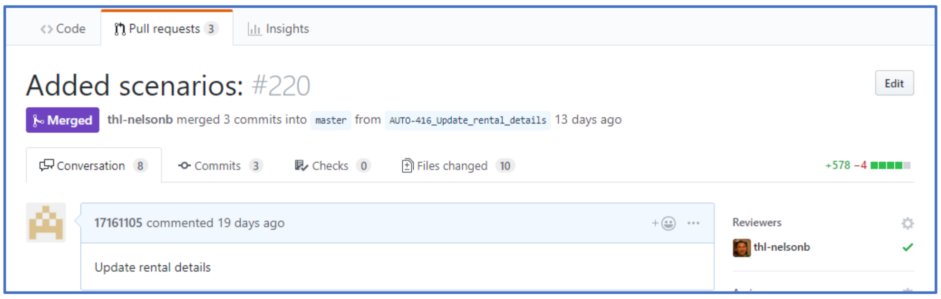
```
{
    [Binding]
    class UpdateContentSteps : Steps
    {
        [Given(@"I have an existing content")]
        public void GivenIHaveAnExistingContent()
        {
            Given("I am on the landing page");
            Given("I click login on the landing page");
            And("I fill in the username");
            And("I fill in the password");
            When("I click the login button");

            var dashboardPage = new DashboardPage();
            Assert.IsTrue(dashboardPage.CosmosHomeHeaderExists(), "You are not in the [redacted] dashboard.");
            dashboardPage.ClickContentsTabButton();

            var contentsPage = new ContentsPage();
            Assert.IsTrue(contentsPage.ContentsHeaderExists(), "You are not in the Contents Page");
            Global.contentId = contentsPage.ApiCreateContent();
            Console.WriteLine(Global.contentId);
            Assert.IsTrue(contentsPage.ContentTableEntriesNotEmpty(), "There are no content entries to update");
        }

        [When(@"I change the (.*) of the content")]
        public void WhenIChangeTheOfTheContent(string field)
        {
```

Figure 9. Step Definitions for update contents



Added scenarios: #220

Merged thl-nelsonb merged 3 commits into master from AUTO-416_Update_rental_details 13 days ago

Conversation 8 Commits 3 Checks 0 Files changed 10 +578 −4

17161105 commented 19 days ago

Update rental details

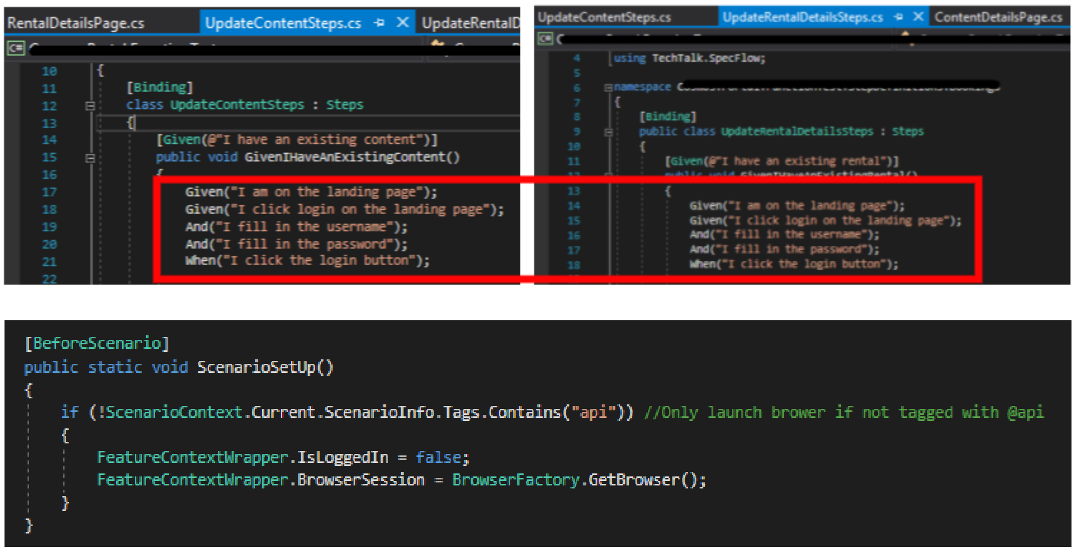
Reviewers
thl-nelsonb ✓

Figure 10. Pull request

4.3.2. Sprint 2

This sprint started with the identification of a common step in all the test scenarios. It has been found that the login steps are common in the existing test scenarios within their step definition classes as shown in Figure 11, including the scripts that were created in Sprint 1. Additionally, it has also been found that browser instantiation is done before the execution of each test scenario through the SpecFlow “Before Scenario” attribute in its Hooks class as shown in Figure 11. These steps have been found to cause unnecessary additional runtimes because they were not part of the tests but were only necessary to be set up before the start of the tests. These steps were removed from the scenario and were handled in the “Before Feature” attribute in the Hooks Class as shown in Error! Reference source not found. Additional codes were created in the “Before Feature” attribute in the Hooks Class so that only one browser instance is created and only one login is performed when running multiple test scenarios in a single test run. An additional check for an existing browser instance and login was also added to catch any unexpected browser instance disposal and logouts during the test run. This was also done to skip browser instantiation and log in when transitioning from one feature class to another during the test run. It was found that the login steps were not available for modification because they were written in a class within an internal assembly file made by GetRV testers to consolidate all common steps.

Test scripts created in Sprint 1 were retrofitted to use this optimisation. The codes highlighted in Figure 11 were removed from each of the tests’ Test Step Definitions classes. The review and feedback process proceeded similarly with Sprint 1.



[redacted]

```
{
    [Binding]
    class UpdateContentSteps : Steps
    {
        [Given(@"I have an existing content")]
        public void GivenIHaveAnExistingContent()
        {
            Given("I am on the landing page");
            Given("I click login on the landing page");
            And("I fill in the username");
            And("I fill in the password");
            When("I click the login button");
```

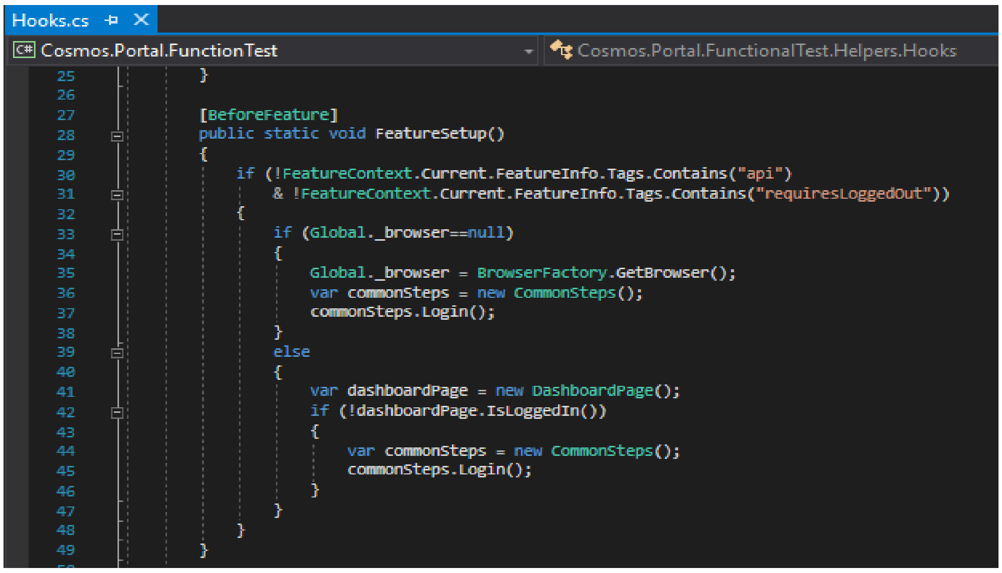
[redacted]

```
using TechTalk.SpecFlow;

namespace [redacted]
{
    [Binding]
    public class UpdateRentalDetailsSteps : Steps
    {
        [Given(@"I have an existing rental")]
        public void GivenIHaveAnExistingRental()
        {
            Given("I am on the landing page");
            Given("I click login on the landing page");
            And("I fill in the username");
            And("I fill in the password");
            When("I click the login button");
```

```
[BeforeScenario]
public static void ScenarioSetUp()
{
    if (!ScenarioContext.Current.ScenarioInfo.Tags.Contains("api")) //Only launch brower if not tagged with @api
    {
        FeatureContextWrapper.IsLoggedIn = false;
        FeatureContextWrapper.BrowserSession = BrowserFactory.GetBrowser();
    }
}
```

Figure 11. Common Steps


Hooks.cs
Cosmos.Portal.FunctionTest
Cosmos.Portal.FunctionalTest.Helpers.Hooks

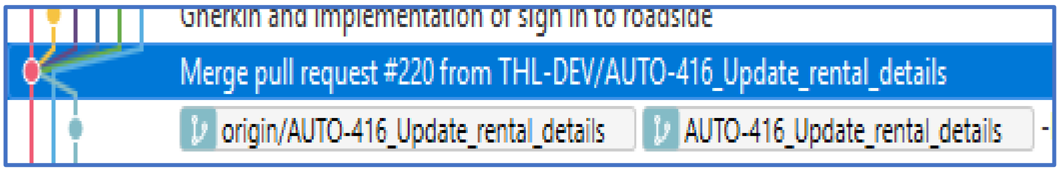
```
        }

        [BeforeFeature]
        public static void FeatureSetup()
        {
            if (!FeatureContext.Current.FeatureInfo.Tags.Contains("api")
                & !FeatureContext.Current.FeatureInfo.Tags.Contains("requiresLoggedOut"))
            {
                if (Global._browser==null)
                {
                    Global._browser = BrowserFactory.GetBrowser();
                    var commonSteps = new CommonSteps();
                    commonSteps.Login();
                }
                else
                {
                    var dashboardPage = new DashboardPage();
                    if (!dashboardPage.IsLoggedIn())
                    {
                        var commonSteps = new CommonSteps();
                        commonSteps.Login();
                    }
                }
            }
        }
```

Figure12. Hooks Class

4.4. Release Phase

There were some changes implemented in the code, those changes were merged to the master branch provided there were no code conflicts between the working branch and master branch. In cases of code conflicts, these were first resolved before merging. Figure 13 shows a sample of a working branch merged to the master branch using Source Tree.



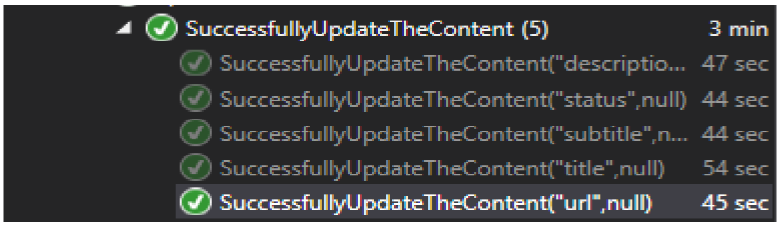
Gherkin and implementation of sign in to roadside
Merge pull request #220 from THL-DEV/AUTO-416_Update_rental_details
origin/AUTO-416_Update_rental_details AUTO-416_Update_rental_details

Figure 13. Source Tree merge to master

5. Results

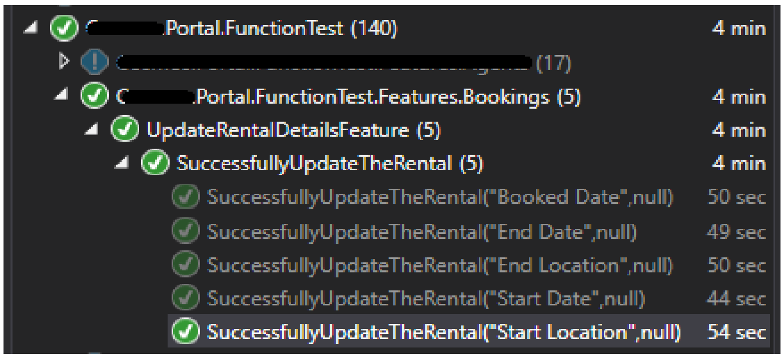
The following are the results of this project:

1. Test results for the two new test cases. Figure 14 shows the results and execution time of the completed “Successfully update the content” test shown in the test runner. Figure 15 shows the result and the execution time of the completed “Successfully update the rental” test shown in the test runner. Table shows a table of consolidated test results of both test runs.



```
SuccessfullyUpdateTheContent (5)                  3 min
  SuccessfullyUpdateTheContent("descriptio...     47 sec
  SuccessfullyUpdateTheContent("status",null)     44 sec
  SuccessfullyUpdateTheContent("subtitle",n...    44 sec
  SuccessfullyUpdateTheContent("title",null)      54 sec
  SuccessfullyUpdateTheContent("url",null)        45 sec
```

Figure 14. Success fully Update The Content Test Result



```
◢ ✓ [redacted].Portal.FunctionTest (140)                                   4 min
  ▷ ! [redacted] (17)
  ◢ ✓ C[redacted].Portal.FunctionTest.Features.Bookings (5)                4 min
    ◢ ✓ UpdateRentalDetailsFeature (5)                                     4 min
      ◢ ✓ SuccessfullyUpdateTheRental (5)                                  4 min
          ✓ SuccessfullyUpdateTheRental("Booked Date",null)               50 sec
          ✓ SuccessfullyUpdateTheRental("End Date",null)                  49 sec
          ✓ SuccessfullyUpdateTheRental("End Location",null)              50 sec
          ✓ SuccessfullyUpdateTheRental("Start Date",null)                44 sec
          ✓ SuccessfullyUpdateTheRental("Start Location",null)            54 sec
```

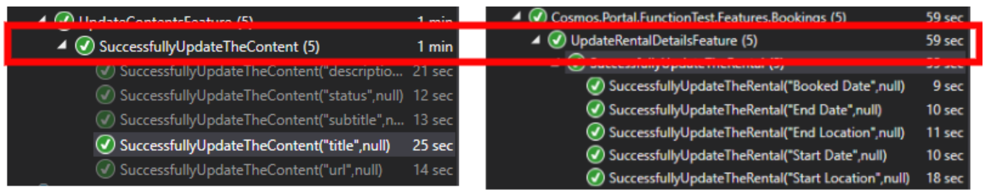
Figure 15. Successfully Update The Rental Test Result

Table 8. Consolidated Test Results (showing un-optimised test scripts runtimes)

Test	Status	Runtime (h:mm:ss)
SuccessfullyUpdateTheContent	Passed	0:03:00
SuccessfullyUpdateTheRental	Passed	0:04:00

2. Comparison of runtimes between un-optimised and optimised test scripts. Run times of un-optimised test scripts are shown in Table. Figure 16 shows the run times of the test scripts after the optimisation has been applied

3.



```
◢ ✓ UpdateContentsFeature (5)                             1 min
  ◢ ✓ SuccessfullyUpdateTheContent (5)                    1 min
      ✓ SuccessfullyUpdateTheContent("descriptio...      21 sec
      ✓ SuccessfullyUpdateTheContent("status",null)      12 sec
      ✓ SuccessfullyUpdateTheContent("subtitle",n...     13 sec
      ✓ SuccessfullyUpdateTheContent("title",null)       25 sec
      ✓ SuccessfullyUpdateTheContent("url",null)         14 sec
```

```
◢ ✓ Cosmos.Portal.FunctionTest.Features.Bookings (5)                  59 sec
  ◢ ✓ UpdateRentalDetailsFeature (5)                                  59 sec
    ◢ ✓ SuccessfullyUpdateTheRental (5)                               59 sec
        ✓ SuccessfullyUpdateTheRental("Booked Date",null)              9 sec
        ✓ SuccessfullyUpdateTheRental("End Date",null)                10 sec
        ✓ SuccessfullyUpdateTheRental("End Location",null)            11 sec
        ✓ SuccessfullyUpdateTheRental("Start Date",null)              10 sec
        ✓ SuccessfullyUpdateTheRental("Start Location",null)          18 sec
```

Figure 16 - Optimised test scripts run times

The percent improvement was computed with this formula:

$$\% \text{ Improvement} = \frac{\text{Before runtime} - \text{after runtime}}{\text{Before runtime}} * 100$$

Table shows a comparison of the test run times and the percent improvement. It is found that there was a substantial improvement in the test run times of the individual scenarios when the test run optimisation was applied.

Table 9. Percent improvement of runtimes

	Runtimes (mm:ss)		
Test	**Before**	**After**	**% Improvement**
Update Content Details	03:00	01:00	**66.67%**
Update Rental Details	04:00	00:59	**75.42%**

6. DISCUSSION

GetRV constantly releases new features or updates to their existing features on their web application, and with these releases, there was a need to validate that the existing features that are not part of the new releases or updates should still be working as expected. GetRV's existing automated regression test does not have updated regression test scenarios to accommodate recent feature releases and updates. To address this, there was a need to align the regression tests to include the new features for regression testing.

As with modern web applications, doing front end regression tests is time consuming even with the help of automated test frameworks. Moreover, adding new test scripts to the automated regression tests will also add to the overall run time of the whole regression test which will negate the benefits of having an automated test framework, to begin with.

The addition of the new additional automation test scripts to cover "Update Content Details" and "Update Rental Details" scenarios add to the overall test coverage of the regression test to increase the confidence of GetRV that when they put out new releases or updates not related to the mentioned scenarios, any failures affecting the scenarios can be caught during the running of the automated regression tests and will not persist to the production environment. Catching these failures before production reduces the costs of fixing them and will prevent any negative feedback from end users.

Applying the approach of Morel et. al., (1979) for removing redundancies in the program greatly reduced the test run times. The test run optimisation was made possible by utilising the Hooks class, a component of SpecFlow to transfer the redundant steps from the start of each test scenario to the start of the test run. From the comparison of the test run times of the optimised and unoptimised test run, it can be deduced that the redundant steps that were removed (browser instantiation and login) account for more than 60 percent or two-thirds of the test run time. These long runtimes may have been caused by slow network connection or hardware and software limitations of the local machine the test is running on. Since all test scenarios performed in the user interface of the web application will require a browser instantiation and log in, the reported improvement on the individual test runs will translate to the same percentage improvement in the overall test run of the regression test. This will help testers to get results quickly and allow for more time to evaluate and plan an appropriate course of action.

This will also align future creation of test scenarios to follow the coding standard (excluding the browser instantiation and login steps from the actual test steps) brought about by the test run optimisation and eliminate the execution of redundant steps.

Due to the feedback and review-oriented approach Scrum methodology, risks of breaking the existing test automation framework were avoided. Furthermore, because of the limited time frame of this project, Scrum has helped in creating a manageable process and breaking down tasks to minimise risks of not achieving those tasks at the end of the project life cycle.

7. CONCLUSION

In conclusion, this project has found two new test scenarios that were not covered by the existing automated regression test and created automated test scripts for them that will be included in the existing automation test framework and in future automated regression test runs. Additionally, this project has filtered the redundant steps that are executed in every scenario and found out that these can be executed once for the whole test run to dramatically reduce both the individual test run times and the overall test run time. Furthermore, the two test scripts created in this project was retrofitted to run with the implemented test run optimisation.

8. RECOMMENDATIONS

Due to time restrictions, only two new user scenarios were converted to automated test scripts to increase the test coverage. The following are the recommendations for GetRV's Automated Regression Test:

- GetRV's testers should review their automated regression test scripts periodically to make sure that new features are covered by the regression tests.
- Checking of the existing test scenarios that may have been skipped or are not run frequently during automated regression test runs and make sure that they are not obsolete.
- Retrofit existing test scripts to run with the test run optimisation by removing the login steps from the step definitions.
- Expose the login methods and steps in the assembly file for modification to avoid code duplication and ease of maintenance.

REFERENCES

[1] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability, 67-120.

[2] Gupta, Harrold, & Soffa. (1997). An Approach to Regression Testing using Slicing. Proceedings The Eighth International Symposium on Software Reliability Engineering (pp. 264-274). Albuquerque, NM, USA, USA: IEEE.

[3] Leung, H., & White, L. (1989). Insights into regression testing (software testing). Proceedings. Conference on Software Maintenance - 1989 (pp. 60-69). Miami, FL, USA, USA: IEEE.

[4] Tarhini, Ismail, & Mansour. (2008). Regression Testing Web Applications. 2008 International Conference on Advanced Computer Theory and Engineering (pp. 902-906). Phuket, Thailand: IEEE.

[5] Tsai, Bai, & Paul. (2001). Scenario-Based Functional Regression Testing. 25th Annual International Computer Software and Applications Conference. COMPSAC 2001 (pp. 496-501). Chicago, IL, USA, USA: IEEE.

[6] Leotta, M., Clerissi, D., Ricca, F., & Spadaro, C. (2013). Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study. 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (pp. 108-113). Luxembourg, Luxembourg: IEEE.

[7] Morel, E., & Renvoise, C. (1979). Global Optimization by Suppression of Partial Redund ancies. Communications of the ACM, 96-103.

[8] Schwaber, K. (1997). SCRUM Development Process. Business Object Design and Implementation (pp.117-134). Austin, Texas: Springer, London.

AUTHORS

Neil Kevin Patalita Arcolas is a graduate of B.S. Electronics and Communications Engineering in the Philip pines and has 2 years background in Software Development. He recently moved to New Zealand to take up Graduate Diploma in Software Testing at AGI Education Limited. This was where he started to explore Soft ware Testing concepts and practices and has taken interest in Functional Test Automation. AGI Education Ltd has opened new opportunities for him in New Zealand and is currently working as a Junior Software Test Analyst in one of the IT companies in New Zealand.

Dr. Shahid Ali is a programme leader of information technology and full-time lecturer at AGI, Auckland, New Zealand. He has completed his Doctorate and master's degree from New Zealand from UIT and AUT respectively. He has published number of research papers in ensemble learning. His expertise and research interests include classification machine learning, data mining, ensemble learning and knowledge discovery.