Pontifical Catholic University of Rio Grande do Sul
TECHNICAL SCHOOL
POSTGRADUATE PROGRAM IN COMPUTER SCIENCE

AN EXAMPLE OF AQUILA USAGE: A DSL FOR MODEL BASED TESTING IN AGILE ENVIRONMENTS

Aline Zanin, Avelino Francisco Zorzo, Henry Cabral Nunes

Technical Report 086

March 2018

AN EXAMPLE OF AQUILA USAGE: A DSL FOR MODEL-BASED TESTING IN AGILE ENVIRONMENTS

Aline Zanin, Avelino Francisco Zorzo, Henry Cabral Nunes
Technology School - PUCRS - Porto Alegre, RS, Brazil
Technical Report number 86

***Resumo*—The application of the Model-based Testing (MBT) technique can bring several benefits to software quality. Usually MBT is applied mostly in traditional software development lifecycle models, for example Waterfall. Wherein few works explore its application in the context of agile teams. Thus, this work presents an example of usage of a Domain Specific Language (DSL), called Aquila, which was designed for modeling functional tests in agile projects. Aquila is an extension of the Domain Specific Language Gherkin, where new keywords related to functional tests are added to allow automated generation of test scripts. This technical report describes, briefly, the Aquila language and how it was applied to a online store system.**

Keywords*—*Software Test, Automation Software Testing, DSL, Áquila, Agile Methods, MBT, Gherkin, Aquila

I. Introduction and Motivation

The increasing search for quality improvement in products and services provided to customers, makes companies seek to optimize their processes and tools to make it possible to serve this purpose. In this sense, the techniques used to software tests also need to be constantly updated and improved.

Among these techniques is Model-based Testing (MBT). MBT focus on the creation of test artifacts and on the reuse of the models created by development teams to perform other activities, for example, requirement analysis and specification, systems analysis and development.The MBT technique adds several benefits to companies, being among the main ones: to facilitate the maintenance of the test artifacts, better traceability between system requirements and test artifacts, and better visualization of requirements [1].

However, although the benefits of MBT are already known and diffused, few studies present the application of this technique in an agile software development context. This is because with the use of MBT in agile context some new challenges emerge.These challenges mainly concern the management of time and resources demanded for the creation of models and the fact that they are built on separate tools from traditional development tools [2][3][4].

These challenges occur specifically in the context of agile software development because teams work with short cycles of continuous deliveries. This work structure requires that the time of creation of the models is reduced so that there is time to execute the tests before the delivery to the customer. Furthermore, agile development methodologies are based on a manifesto for the development of agile software [5], and in the document, principles of communication, interaction and collaborative work are essential, being more important then extensive documentation, individual responsibilities and contractual bureaucracy.

However, while reducing bureaucracy and optimizing documentation, it is necessary to use some strategy to document system requirements. However, in order to enable this collaborative environment between the members of the development team and those with the client, it is necessary to establish a form of communication where everyone can understand what will be developed in the software. To this end it is common in agile development teams to use user stories for requirements documentation [6].

With the use of user stories, the client can describe their expectations regarding the software using natural language. From these user stories, systems development teams can derive scenarios that are used as Living Documentation generating artifacts used during the development process. An example of a technique that uses Living Documentation is Behavior Driven Development (BDD), which, through the Gherkin Domain Specific Language (DSL), describes in detail the scenarios that compose each of the user stories, maintaining the natural language and generating the methods that guide tests automation.

Nevertheless, Behavior Driven Development (BDD) does not automatically provide the specification of the steps that comprise the test scripts, and it is necessary for the tester to manually create the representation of the steps of each action that must be performed in the system to test each scenario. In addition, the BDD technique does not present test sequences to the tester, which requires the creation of sequences according to their knowledge of the system. Thus, allying the MBT and BDD techniques can add advantages to agile teams. One of the alternatives would be to use a DSL.

DSL are languages designed to meet the needs of a specific domain and make use of vocabularies that are usual in this domain. In this context, we show in this technical report an example of usage of a DSL that can be applied in agile environments. Aquila is used for the representation of test scenarios through semi-natural language with an addition of test and system information, making possible the automated creation of models for MBT execution.

This technical report is organized as follows. Section II shows a brief description of the Aquila language. Section III

shows the use of Aquila to model an online store system.

II. ÁQUILA - A DSL FOR AGILE SOFTWARE DEVELOPMENT

Aiming to propose a facilitated alternative for MBT, and considering a main focus in teams that work with agile methodologies of software development, we defined a new DSL called Aquila. This DSL proposes to extend the Gherkin DSL to automatically create models from the scenarios written in a semi-natural language. This was achieved by aggregating keywords that represent system information. Among these information are references to the actions that need to be performed in the system to execute a test, for example, correct filling of the fields present in the graphical interface of certain functionality.

With the automated generation of models from the Aquila scenarios, it becomes possible to automate the creation and to facilitate the maintenance of test scripts. Hence, the maintenance is facilitated through the created traceability, where, the change of a scenario is reflected in the alteration of all the scripts that implement it.

An automated generation of test scripts from models is an already existing process applied by several researchers [7] [8] [9]. However, these works consider manual creation of models or generation of models from source code, and do not consider generation of models from software requirements.

The generation of test scripts from source code has drawbacks since it can lead to tendentious testing, and one of the seven principles of software testing is to perform tests based on system requirements and not based on the implemented system [10]. However, the above mentioned disadvantage is not applied to regression tests where it is intended to ensure that a system remains in operation after a code change [10]. The manual creation of models, in the process of agile development, is often cited as one of the main points that hinders the implementation of MBT, mainly due to short cycles of continuous delivery applied in agile methodologies [11].

Different from the above mentioned works, our proposal allows the generation of test scripts based exclusively on the specified requirements, the established system standards and the set of inputs and outputs expected for the system. Therefore, the generated test artifacts represent what the system must do and not what the implemented system does, thus increasing the efficiency of the tests.

The basic features of Aquila are:

- Aquila is designed as an extension of Gherkin.
- New keywords (called tags) were added in Aquila.
- The Aquila DSL is used following a structure of files predefined in the development project.
- In order to establish traceability between requirements and model:
  1) The GIVEN word will represent the start model of a feature.
  2) The WHEN word will represent the creation of a new activity in a model.
  3) The AND word, within the WHEN clause, will represent the creation of a new activity in the model.

Tabela I. AQUILA DSL KEYWORDS

Tag	Definition	Where it is used
< >	Used to enter a URL	GIVEN
{ }	Used to reference a scenario in another scenario	GIVEN
click[field]	Represents the action of clicking on a specific field or link	WHEN
put[field]	Represents the action of inserting values in a specific field	WHEN
use-valid-data	Represents that all fields present on the feature under test and listed in the input table will receive valid values	WHEN
dont-fill-out[field]	represents that no values will be entered in a specific field	WHEN
checked[field]	Represents the selection of an option in a checkbox	WHEN
choose[field]	Represents the selection of an option in a radio button	WHEN
select-data[field]	Represents the selection of an option in a list	WHEN
mouse-over[field]	Represents the placement of the mouse over a given field	WHEN
enable[field]	Represents verification if a field is active	THEN
disable[field]	Represents verification if a field is not active	THEN
showed[field]	Represents checking for the presence of a particular word or phrase on the screen	THEN
opened[url]	Represents checking the correct opening of a page	THEN
showed-title[word]	Represents page title verification	THEN
showed[value]in[field]	Represents checking for the presence of a value in a specific field	THEN

  4) All Aquila scenarios that require data input or value selection, will use a table to input values for each entry fields. This table is inherited from Gherkin and can be seen in Section III-A.
  5) The word USE-VALID-DATA, in the WHEN clause, will represent the creation of one activity for each field present in the interface and specified in the input table.
  6) When using USE-VALID-DATA and the application has fields that allow selection (select, radio button or checkbox), the value specified in the input table must be selected accordingly. For example, for a radio button field with the “female” and “male” options, if the tester wants the masculine option to be selected, the tester must specify “male” in the input table.
  7) When the scenario has more than one input to be exercised, Aquila should consider each entry as a different branch in the model and consequently create a script for each entry.
  8) Negative Test Cases (for example, not filling in required fields) should be written in specific scenarios for this purpose.
  9) In order to perform negative tests where the negativity is due to the use of invalid entries, the put[] tag must be used in Aquila and in the input table invalid entries have to be used.

10) The situations described in the THEN clause are converted into activities in the model, however the scripts generated by them will always have the purpose of verifying the results generated by the test. The THEN clause will be complemented by: enable []; disable []; showed []; opened []; showed-title []; showed []in [].

Table I shows the selected new keywords that were included in Aquila. The table shows: in the left column, the keywords (tags); in the middle column, the behavior represented by each of the tags; and, in the right column, the place where each of them can be used.

The previous mentioned TAGS, together with the TAGS used in Gherkin (GIVEN, WHEN, THEN), are inserted in the scenarios in order to represent the actions that can be performed in the system and the expected results for each one of them.

A. Aquila DSL Utilization

The flow of Aquila usage can be seen in Figure 1. This figure represents the six steps that can be accomplished using Aquila.

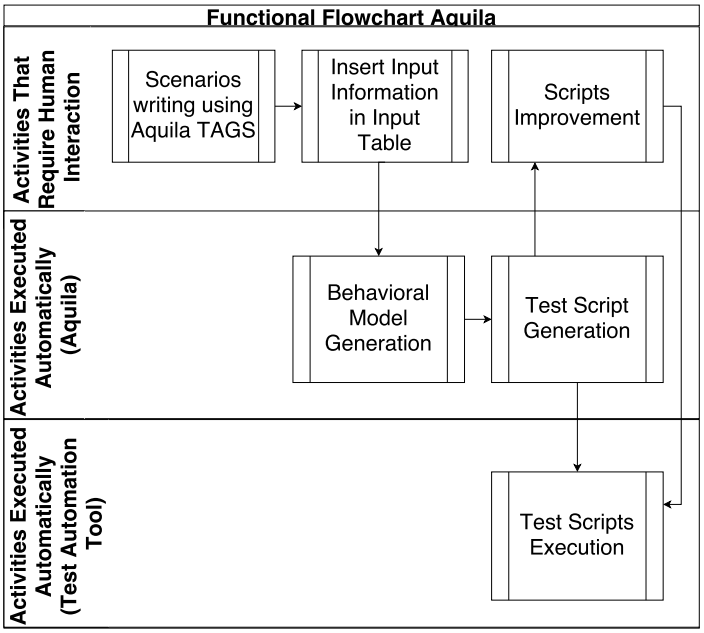

Figura 1. Aquila Functional Flowchart

1) The first step is the writing of Aquila scenarios. In this step, the insertion of the keywords that compose the DSL to represent the system information is performed.
2) The second step is responsible for inserting information about the fields present in the interface of the system being tested and the inputs to which these fields are submitted. This is performed through a table that contains the names of the fields and their respective entries.
3) The third step is the generation of behavioral models. In this step each of the lines of a WHEN clause and their respective ANDs are reflected in an activity in the activity diagram. There are two exceptions to this rule, the first exception is when using the use-valid-data tag because this tag will indicate an activity for each of the fields that is specified in the input table. The second exception is when for some field two inputs are specified, in this case two parallel activities will be generated, one for each input.
4) The fourth step is the test script generation. In this step, based on the information of tests written in the scenarios and in the generated models, the generation of the scripts is performed automatically. In this work we use specific code for Seleniun Webdriver tool, however, Aquila can be used for any other automation tool.
5) The fifth step is to improve the generated scripts. This improvement is necessary in cases where a test engineer uses some field name different from the one used in the application, to correct some mistake in the scenarios, or to create a script for an action that is not completely covered by Aquila. In these cases a partial script is generated, and the test engineer needs to complete the generated script.
6) The sixth step is the execution of the generated scripts. At this stage Aquila uses test automation tools. The chosen tool should be compatible with the tool used for test scripts generation.

III. Exemple of Use: an online store system

In order to verify Aquila in an environment close to a real one, we used a didactic virtual store, available free in the internet to be used by people who are learning test automation. The chosen online store is available at http://store.demoqa.com/. This fake store sells electronic products, more specifically smartphones. In this store it is possible to select products according to categories, add these products to the shopping cart, carry out user registration, log in and log out and finalize the purchase of the product. In this section, we show how to generate test scripts to test the application that is running on that fake store. The next sections show how to use Aquila for scenario creation, insert data in a table, models generation, and test script generation.

Creating Scenarios

This is the main step when using Aquila, because the correct creation of the scenarios will influence directly the generated scripts. In this step, the test and system information that extends Gherkin are inserted. Figure 2 shows an example of Aquila code to describe test case scenarios.

A. Table Insertion - sample

In order to be able to generate complete test scenarios, it is necessary to enter the name of the fields that will be manipulated in the respective scenario and the data that will be used for each field.

This entry is derived from Gherkin, however, Aquila also includes the representation of values that should be selected in the selection fields (for example, checkbox, radio button, and select). It is important to mention that field types in Aquila and in the system have to be the same, otherwise test will fail.

Scenario: 3- Test Add To Cartg
Given < http://store.demoqa.com>
When I **mouse-over**[product Category]
And I **click**[imac]
And I **click**[add to card]
Then Showed[Item has been added to your cart!]

Scenario: 5- Shopping Cart Quantity Update
Given {Shopping Cart}
When I **put**[quantity]
And I **click**[Update]
Then showed[$750.00]
And showed[$150,00]**in**[price]

Scenario: 7- Shopping Cart Calculate Shipping
Given {Shopping Cart}
When I **click**[Continue]
And I **select-data**[country]
And I **select-data**[state]
And I click[calculate]
Then showed[Shipping Price]

Scenario: 8- New user registration
Given {Test Add To Card}
When I **click**[checkout]
And I **click**[continue]
And I **use-valid-data**
And I **dont-fill-out**[username]
And I **dont-fill-out**[password]
And I **click**[purchase]
Then showed[Sub-Total]
And showed[Magic Mouse]

Scenario: 9- Checkout Sign In
Given {Shopping Cart Calculate Shipping }
When I **put**[Username]
And I **put**[Password]
And I **click**[login]
Then showed[Login sucess]

Figura 2. Scenarios

Scenario: 7- Shopping Cart Calculate Shipping
Given {Shopping Cart}
When I **click**[Continue]
And I **select-data**[country]
|**Country**|**State** |
|Brasil |Rio Grande do Sul|
And I **select-data**[state]
And I click[calculate]
Then showed[Shipping Price]

Scenario: 8- New user registration
Given {Test Add To Card}
When I **click**[checkout]
And I **click**[continue]
And I **use-valid-data**

country	**state**	**email**	**FirstName**
Brasil	Rio Grande do Sul	al@test.com	Aline
EUA	California	la@test.com	

LastName	**Address**	**City**	**Postal Code**	**Phone**
Zanin	xyxxzzy	yklj	900789450	41888776553
@345			13456	1-866-333-4606

And I **dont-fill-out**[username]
And I **dont-fill-out**[password]
And I **click**[purchase]
Then showed[Sub-Total]
And showed[Magic Mouse]

Scenario: 9- Checkou Sign In
Given {Checkout Calculate Shipping Price}
When I **put**[Username]
|**username**|
|y6usname|
And I **put**[Password]
|**Password**|
|pas1235rd|
And I **click**[login]
Then showed[Login sucess]

Figura 3. Tables

The Aquila DSL tags that need to have explicit inputs in the input table are as follows: put[]; use-valid-data[]; checked[]; choose[]; select-data[]. In the example for the e-commerce system, input table were used in scenarios 7, 8 and 9 as can be seen in Figure 3.

B. Model generation

For each scenario, Aquila DSL will create a behavioral model, in this example a UML activity diagram. Each action described in the GIVEN, WHEN and AND tags (when AND is associated with the WHEN clause), and each result described in the THEN and AND tags (when AND is associated with THEN clause) will reflect on an activity in the model. The exception to this is the USE-VALID-DATA tag that generates not only one activity but an activity for each field present on the screen.

All models generated for the scenarios shown in Figure 2 can be seen in Appendix A. Figure 6 shows the model for the ” 9 - New User Registration ” scenario, which represents the registration of a new user into the system. As can be seen in the figure, in some cases there two flows, which represent the possible entries as shown in Figure 3. For each flow, a test script is generated.

C. Test Script Generation

From the generated models, the test scripts are generated, and for each scenario a script is generated except in cases where, for some field, the use of two distinct inputs is required, and therefore, a test script is generated for each input.

Figure 5 shows a snipet of a code generated for the Selenium Webdriver tool. This tool was chosen because it is widely used in the market. However Aquila can be used with other tools, the same Aquila code can be used to generate scripts for different tools or languages. Figure 5 shows the code for the “New

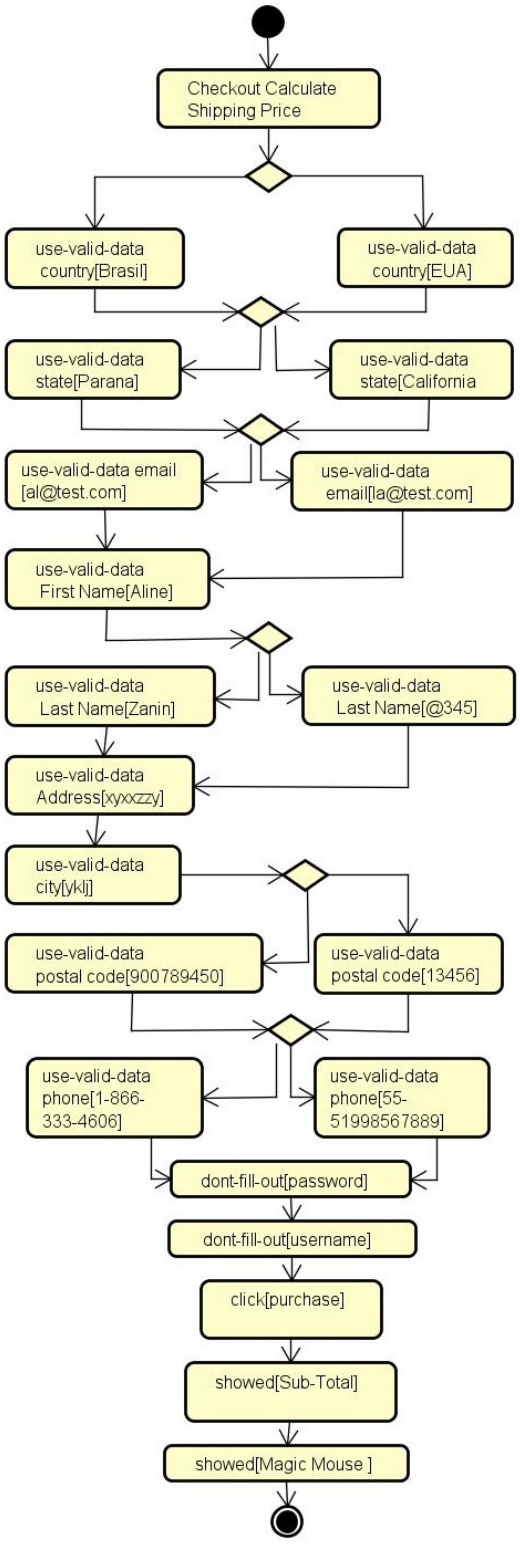

Figura 4. New User Registration Model

```
1       @Test
2       public void newUserRegistration() {
3
4       /*Given < http://store.demoqa.com> */
5         driver.get("http://store.demoqa.com/");
6       /*When I mouse-over[product Category]*/
7         WebElement element = driver.findElement(
              By.linkText("Product Category"));
8         Actions action = new Actions(driver);
9         action.moveToElement(element).build().
              perform();
10      /*And I click[imac]*/
11        driver.findElement(By.linkText("iMacs"))
              .click();
12      /*And I click[add to card]*/
13        driver.findElement(By.name("Buy")).click
              ();
14      /*And I click[checkout]*/
15        driver.findElement(By.partialLinkText("
              Checkout")).click();
16      /*And I click[continue]*/
17        driver.findElement(By.partialLinkText("
              Continue")).click();
18      /* use-valid-data country*/
19        Select country = new Select(driver.
              findElement(By.name("country")));
20        country.selectByVisibleText("Brasil");
21      /* use-valid-data state*/
22          WebElement state = driver.findElement(
                By.name("state"));
23          state.sendKeys("Rio Grande do Sul");
24      /* use-valid-data email*/
25          WebElement email = driver.findElement(
                By.name("email"));
26          state.sendKeys("al@test.com");
27      /* use-valid-data firsName*/
28          WebElement firsName = driver.
                findElement(By.name("firsName"));
29          state.sendKeys("Aline");
30      /* use-valid-data LastName*/
31          WebElement LastName = driver.
                findElement(By.name("LastName"));
32          state.sendKeys("Zanin");
33      /* use-valid-data Address*/
34          WebElement Address = driver.
                findElement(By.name("Address"));
35          state.sendKeys("xyxxzzy");
36      /* use-valid-data city*/
37          WebElement city = driver.findElement(
                By.name("city"));
38          city.sendKeys("yklj");
39      /* use-valid-data PostalCode*/
40          WebElement PostalCode = driver.
                findElement(By.name("PostalCode"));
41          PostalCode.sendKeys("900789450");
42
43  ...
44      }
```

Figura 5. Generated Code for “New User Registration” scenario

User Registration" scenario. Scripts generated for the scenarios shown in Figure 2 can be seen in Appendix B.

IV. Conclusion

This work presented a way to work with model-based testing in agile software development. To show how this could be achieved we applied Aquila in the development of an online store system. This was applied in an agile development environment. Aquila extends Gherkin with new keywords to represent system and test actions. This example will be used during the validation of Aquila in a Focus Group evaluation methodology.

Referências

[1] I. K. El-Far and J. A. Whittaker, "Model-based software testing," *Encyclopedia of Software Engineering*, 2001.

[2] M. Katara and A. Kervinen, "Making model-based testing more agile: a use case driven approach," in *Proceedings of the Haifa Verification Conference*, 2006.

[3] D. Jalalinasab and R. Ramsin, "Towards model-based testing patterns for enhancing agile methodologies." in *SoMeT*, 2012.

[4] V. Entin, M. Winder, B. Zhang, and S. Christmann, "Introducing model-based testing in an industrial Scrum project," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012.

[5] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," http://www.agilemanifesto.org, 2001, acessado em 03/07/2017.

[6] VersionOne, "State of agile report," http://stateofagile.versionone.com/, 2017, acessado em 11/06/2017.

[7] A. Domingues, E. M. Rodrigues, and M. Bernardino, "Autofun: An automated model-based functional testing tool," in *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. ACM, 2016.

[8] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, "A test generation solution to automate software testing," in *Proceedings of the 3rd International Workshop on Automation of Software Test*, 2008.

[9] J. Lasalle, F. Peureux, and F. Fondement, "Development of an automated mbt toolchain from uml/sysml models," *Innovations in Systems and Software Engineering*, vol. 7, no. 4, 2011.

[10] I. S. T. Q. B. ISTQB, "Syllabi," http://www.istqb.org/downloads/syllabi.html, acessado em 11/06/2017.

[11] V. Entin, M. Winder, B. Zhang, and S. Christmann, "Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach," in *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011.

V. Appendix A - Models

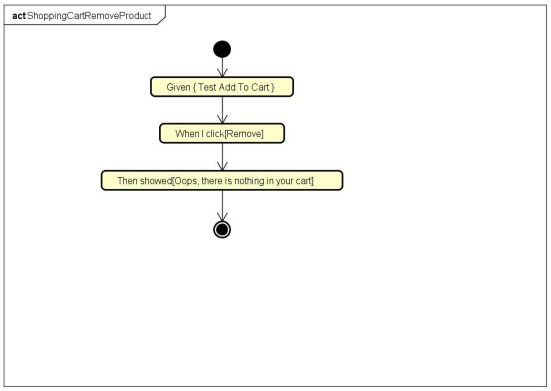

Figura 6. Shopping Cart Remove Product Model

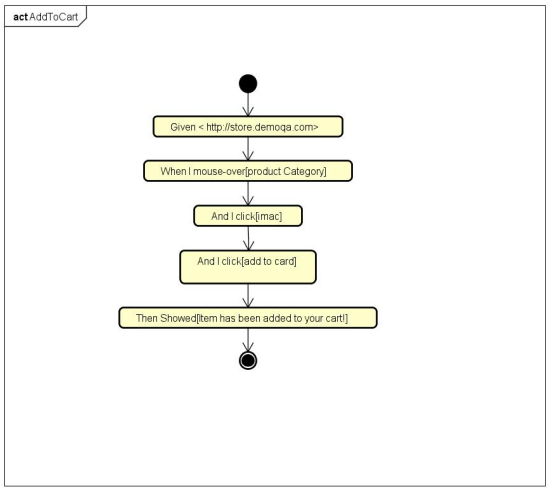

Figura 7. Add To Card Model

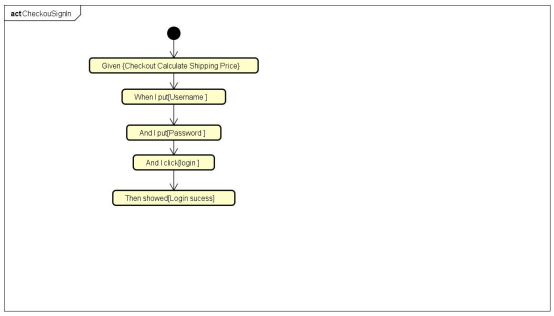

Figura 8. Checkou Sign In Model

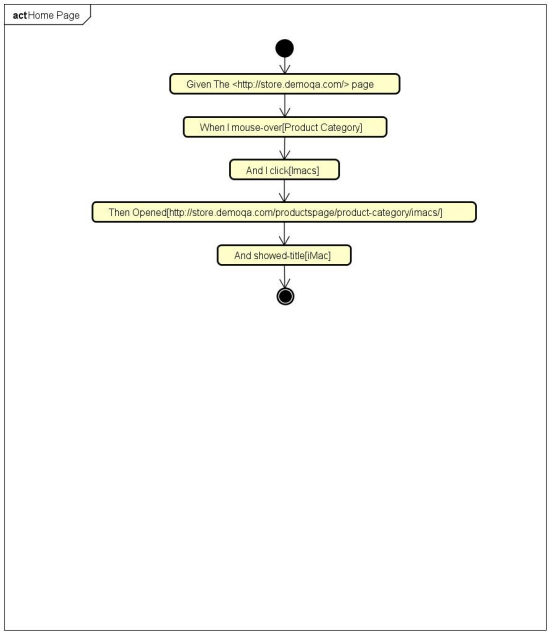

Figura 9. Home Page Model

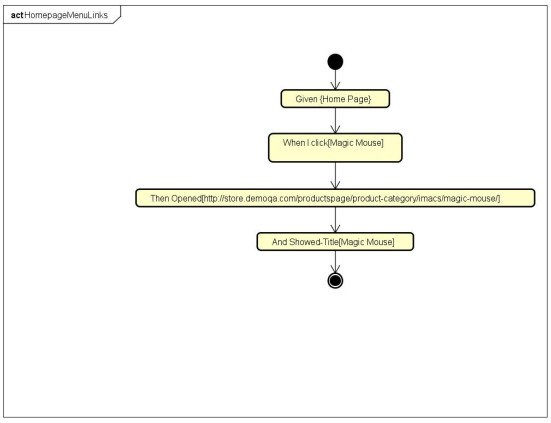

Figura 10. Home page Menu Links Model

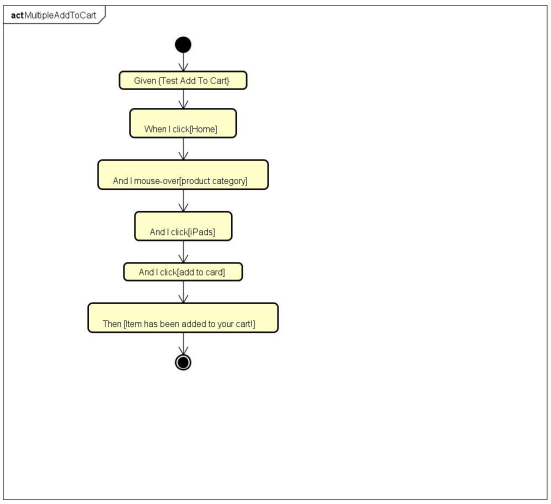

Figura 11. Multiple Add To Cart Model

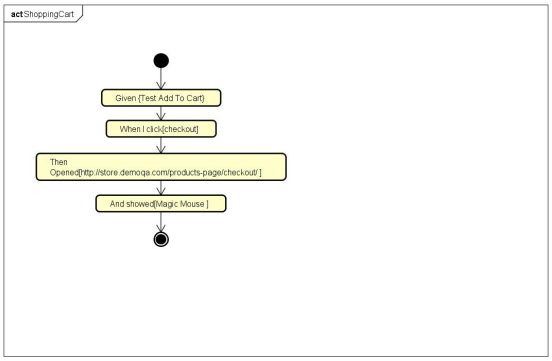

Figura 12. Shopping Cart Model

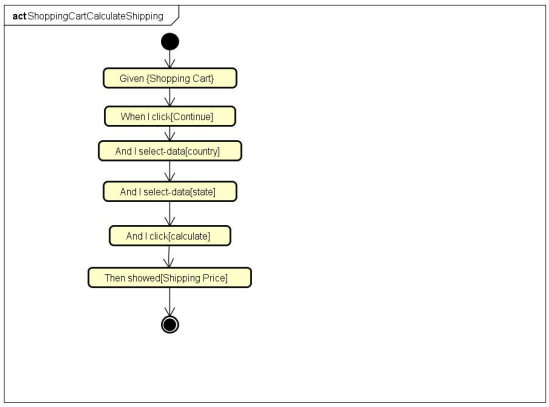

Figura 13. Shopping Cart Calculate Shipping Model

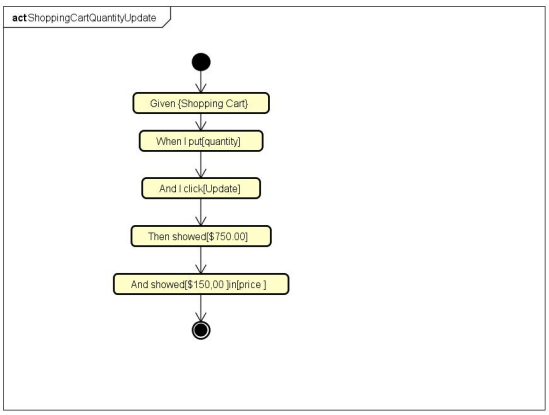

Figura 14. Shopping Cart Quantity Update

VI. Appendix B - Scripts

```java
public class AddToCard {

    WebDriver driver;
    @Test
    public void testAddToCard() {

    /*Given < http://store.demoqa.com>*/
        System.setProperty("webdriver.gecko.
            driver", "C:\\Users\\danie\\
            Documents\\eclipse\\geckodriver.exe
            ");
        driver = new FirefoxDriver();
    /* Given < http://store.demoqa.com> */
        driver.get("http://store.demoqa.com/")
            ;
    /* When I mouse-over[product Category] */
        WebElement element = driver.
            findElement(By.linkText("Product
            Category"));
        Actions action = new Actions(driver);
        action.moveToElement(element).build().
            perform();
    /*  And I click[imac]*/
        driver.findElement(By.linkText("iMacs"
            )).click();
    /*  And I click[add to card]*/
        driver.findElement(By.name("Buy")).
            click();
    /*  Then Showed[Item has been added to
        your cart!] */
        String ItemHasBeenAddedToYourCart =
            driver.getPageSource();
        assertEquals(true,
            ItemHasBeenAddedToYourCart.contains
            ("Item Has Been Added To Your Cart"
            ));



    }

}
```

Figura 15. Add To Card

```java
public class CheckouSignIn {

    WebDriver driver;

    public void testCheckouSignIn() {

        /* Given {Checkout Calculate Shipping
            Price} */
        ShoppingCartCalculateShipping sccs =
            new ShoppingCartCalculateShipping()
            ;
        sccs.testShoppingCartCalculateShipping
            ();
        /* When I put[Username ] */
        WebElement Username = driver.
            findElement(By.name("Username"));
        Username.sendKeys("y6usname");
        /* And I put[Password ] */
        WebElement Password = driver.
            findElement(By.name("Password"));
        Password.sendKeys("pas1235rd");
        /* And I click[login ] */
        WebElement login = driver.findElement(
            By.name("login"));
        login.click();
        /* Then showed[Login sucess] */
        String Loginsucess = driver.
            getPageSource();
        assertEquals(true, Loginsucess.
            contains("Login sucess"));
    }
}
```

Figura 16. Checkout Sign In

```java
public class HomePage {

    WebDriver driver;

    @Before
    public void setUp() {

    }

    @Test
    public void TestHomePage() {
        System.setProperty("webdriver.gecko.
            driver", "C:\\Users\\danie\\
            Documents\\eclipse\\geckodriver.exe
            ");
        driver = new FirefoxDriver();
    /* Given < http://store.demoqa.com> */
        driver.get("http://store.demoqa.com/")
            ;
    /* When I mouse-over[product Category] */
        WebElement element = driver.
            findElement(By.linkText("Product
            Category"));
        Actions action = new Actions(driver);
        action.moveToElement(element).build().
            perform();
    /* And I click[imac] */
        driver.findElement(By.linkText("iMacs"
            )).click();
    /* Then Opened[http://store.demoqa.com/
        productspage/product-category/imacs/]
        */
        String url = driver.getCurrentUrl();
        assertEquals("http://store.demoqa.com/
            products-page/product-category/
            imacs/", url);
    /*And showed-title[iMac]*/
        String titulo = driver.getTitle();
        assertEquals(true, titulo.contains("
            iMac"));

    }
}
```

Figura 17. Test Home Page

```java
public class HomePage {

    WebDriver driver;

    @Before
    public void setUp() {

    }

    @Test
    public void TestHomePage() {
        System.setProperty("webdriver.gecko.
            driver", "C:\\Users\\danie\\
            Documents\\eclipse\\geckodriver.exe
            ");
        driver = new FirefoxDriver();
    /* Given < http://store.demoqa.com> */
        driver.get("http://store.demoqa.com/")
            ;
    /* When I mouse-over[product Category] */
        WebElement element = driver.
            findElement(By.linkText("Product
            Category"));
        Actions action = new Actions(driver);
        action.moveToElement(element).build().
            perform();
    /* And I click[imac] */
        driver.findElement(By.linkText("iMacs"
            )).click();
    /* Then Opened[http://store.demoqa.com/
        productspage/product-category/imacs/]
        */
        String url = driver.getCurrentUrl();
        assertEquals("http://store.demoqa.com/
            products-page/product-category/
            imacs/", url);
    /*And showed-title[iMac]*/
        String titulo = driver.getTitle();
        assertEquals(true, titulo.contains("
            iMac"));

    }
```

Figura 18. Home Page

```java
public class HomepageMenuLinks {

    WebDriver driver;

    @Test
    public void TestHomePageMenuLinks() {
        System.setProperty("webdriver.gecko.
            driver", "C:\\Users\\danie\\
            Documents\\eclipse\\geckodriver.exe
            ");
        driver = new FirefoxDriver();
        /*Given {Home Page}*/
        HomePage h = new HomePage();
        h.TestHomePage();
        /*When I click[Magic Mouse]*/
        WebElement MagicMouse = driver.
            findElement(By.name("MagicMouse"));
        MagicMouse.click();

        /*Then Opened[http://store.demoqa.com/
            productspage/product-category/imacs
            /magic-mouse/]*/
        String url = driver.getCurrentUrl();
        assertEquals("http://store.demoqa.com/
            products-page/product-category/
            imacs/", url);
        /*And showed-title[Magic Mouse]*/
        String MagicMouseTitle = driver.
            getTitle();
        assertEquals(true, MagicMouseTitle.
            contains("Magic Mouse"));


    }

}
```

Figura 19. Test Home Page Menu Links

```java
public class MultipleAddToCart {

    WebDriver driver;

    @Test
    public void TestMultipleAddToCart() {

    /* Given {Test Add To Cart} */
        AddToCard addToCard = new AddToCard();
        addToCard.testAddToCard();
    /* When I click[Home] */
        WebElement home = driver.findElement(
            By.name("Home"));
        home.click();
    /* And I mouse-over[product category] */
        WebElement element = driver.
            findElement(By.linkText("Product
            Category"));
        Actions action = new Actions(driver);
        action.moveToElement(element).build().
            perform();
    /* And I click[iPads] */
        WebElement iPads = driver.findElement(
            By.name("iPads"));
        iPads.click();
    /* And I click[add to card] */
        driver.findElement(By.name("Buy")).
            click();
    /* Then [Item has been added to your cart
        !] */
        String ItemHasBeenAddedToYourCart =
            driver.getPageSource();
        assertEquals(true,
            ItemHasBeenAddedToYourCart.contains
            ("Item Has Been Added To Your Cart"
            ));

    }

}
```

Figura 20. Test Multiple Add To Cart

```java
public class ShoppingCart {

    WebDriver driver;
    @Test
    public void TestShoppingCart() {

        /*Given {Test Add To Cart}*/
        AddToCard addToCard = new AddToCard();
        addToCard.testAddToCard();

        /*When I click[checkout]*/
        driver.findElement(By.partialLinkText(
            "Checkout")).click();

        /*Then Opened[http://store.demoqa.com/
            products-page/checkout/ ]*/
        String url = driver.getCurrentUrl();
        assertEquals("http://store.demoqa.com/
            products-page/checkout/", url);

        /*And showed[Magic Mouse ]*/
        String MagicMouse = driver.
            getPageSource();
        assertEquals(true, MagicMouse.contains
            ("Magic Mouse"));
    }
}
```

Figura 21. Shopping Cart

```java
public class ShoppingCartCalculateShipping {

    WebDriver driver;

    @Test
    public void
        testShoppingCartCalculateShipping() {
        /* Given {Shopping Cart} */
        ShoppingCart sc = new ShoppingCart();
        sc.TestShoppingCart();

        /* When I click[Continue] */
        driver.findElement(By.name("Continue")
            ).click();
        /* And I select-data[country] */
        Select country = new Select(driver.
            findElement(By.name("country")));
        country.selectByVisibleText("Brasil");
        /* And I select-data[state] */
        WebElement state = driver.findElement(
            By.name("state"));
        state.sendKeys("Rio Grande do Sul");
        /* And I click[calculate] */
        driver.findElement(By.name("calculate"
            )).click();
        /* Then showed[Shipping Price] */
        String ShippingPrice = driver.
            getPageSource();
        assertEquals(true, ShippingPrice.
            contains("Shipping Price"));
    }
}
```

Figura 22. Shopping Cart Calculate Shipping

```java
public class ShoppingCartQuantityUpdate {

    WebDriver driver;

    @Test
    public void test() {
        /* Given {Shopping Cart} */
        ShoppingCart sc = new ShoppingCart();
        sc.TestShoppingCart();

        /* When I put[quantity] */
        WebElement quantity = driver.
            findElement(By.name("quantity"));
        quantity.sendKeys("5");
        /* And I click[Update] */
        WebElement Update = driver.findElement
            (By.name("Update"));
        Update.click();
        /* Then showed[$750.00] */
        String showed = driver.getPageSource()
            ;
        assertEquals(true, showed.contains("
            750.00"));

        /* And showed[$150,00 ]in[price ] */
        WebElement price = driver.findElement(
            By.name("price"));
        String price2 = price.getAttribute("
            value");
        assertEquals(true, price2.contains("
            150,00"));
    }

}
```

Figura 23. Shopping Cart Quantity Update

```java
public class ShoppingCartRemoveProduct {

    WebDriver driver;
    @Test
    public void testShoppingCartRemoveProduct
        () {

    /*Given { Test Add To Cart }*/
    AddToCard addToCard = new AddToCard();
    addToCard.testAddToCard();
    /*When I click[Remove]*/
    WebElement remove = driver.findElement(By.
        name("remove"));
    remove.click();
    /*Then showed[Oops, there is nothing in
        your cart]*/
    String OopsThereIsNothingInYourCart =
        driver.getPageSource();
    assertEquals(true,
        OopsThereIsNothingInYourCart.contains("
        Oops, There Is Nothing In Your Cart"));

    }

}
```

Figura 24. Shopping Cart Remove Product