ECSE428
March 5th, 2017

Assignment B

Summary of Deliverables:

Story Statement

Story:
As an Amazon consumer,
I would like to be able to checkout the products in my cart
so I can purchase them.

Normal Flow: Checking out after adding a new product to shopping cart
Given I am on a Amazon product page
When I press "Add to cart"
Then the product should exist in my shopping cart
And the checkout button exists

Alternate Flow: Checking out after adding an existing product to shopping cart
Given I am on a Amazon product page
And I have the same product that already exists in my shopping cart
When I press "Add to cart"
Then the product should exist in my shopping cart
And the quantity of the product should be equal to two
And the checkout button exists

Error Flow: Checking out after removing an existing product from shopping cart
Given I am on my current shopping cart
And I have a product that exists in my shopping cart
When I press "Delete"
Then the product should no longer exist in my shopping cart
And the checkout button does not exist

Error Flow: Checking out with an empty shopping cart
Given I am on my current shopping cart
And my shopping cart is empty
Then there is nothing to delete from the shopping cart
And the checkout button does not exist

Environment Description

General Setup

Both step definitions and fixtures for Cucumber and FitNesse were created in the same project using Java 8 and Maven 3 with IntelliJ Ultimate IDE. In addition, Selenium v3.1 was used for web automation and was essential to testing the features of Amazon. Specifically with Selenium, the Chrome Web Driver was used for all acceptance tests. The tests were run with MacOS Sierra v10.12.3. More detail about the source code is available in the later sections of this document.

Cucumber

The dependencies for Cucumber, with version 1.0.11, were installed using Maven. All cucumber tests run locally on the machine through IntelliJ. A feature file was created, *amazon.feature*, which contains all the scenarios. Using IntelliJ, we can test by running the feature file locally and ensuring that the configurations are set accordingly (e.g. Glue is set to *com.kevinnam.cucumber*). More details about installing and running the cucumber-based tests are available in later sections. The following are the descriptions of the maven cucumber dependencies:

```
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-java</artifactId>
  <scope>test</scope>
  <version>1.0.11</version>
</dependency>
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-jvm</artifactId>
  <version>1.0.11</version>
  <type>pom</type>
</dependency>
```

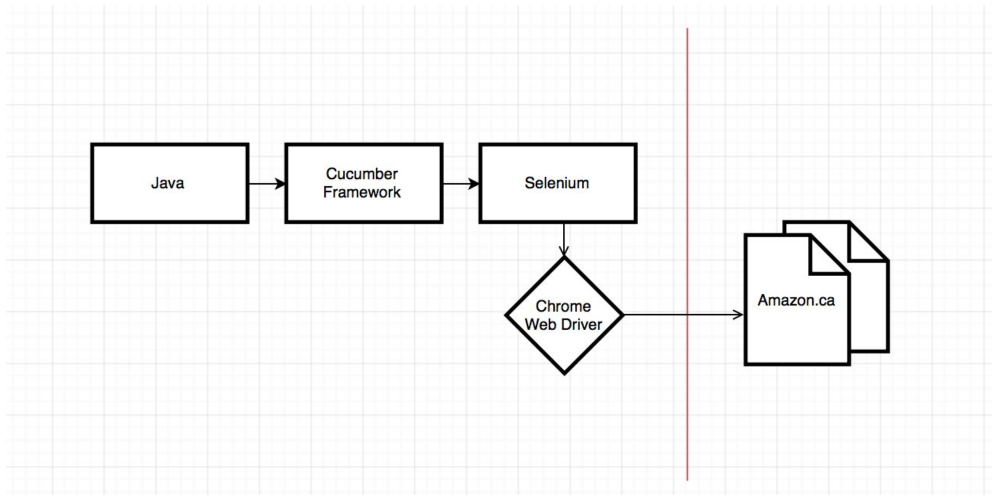
FitNesse

FitNesse version 20161106 was installed and run locally for testing. Specifically, *fitnesse-standalone.jar* was run on port 3000, which gives us access to the FitNesse framework at *localhost:3000*. A Wiki page, *Checking out a product Acceptance Tests* , was created and contains all the acceptance test cases. To run Selenium with FitNesse, it was essential to add a classpath to *selenium-server-standalone-3.1.0.jar*.

Block Diagram

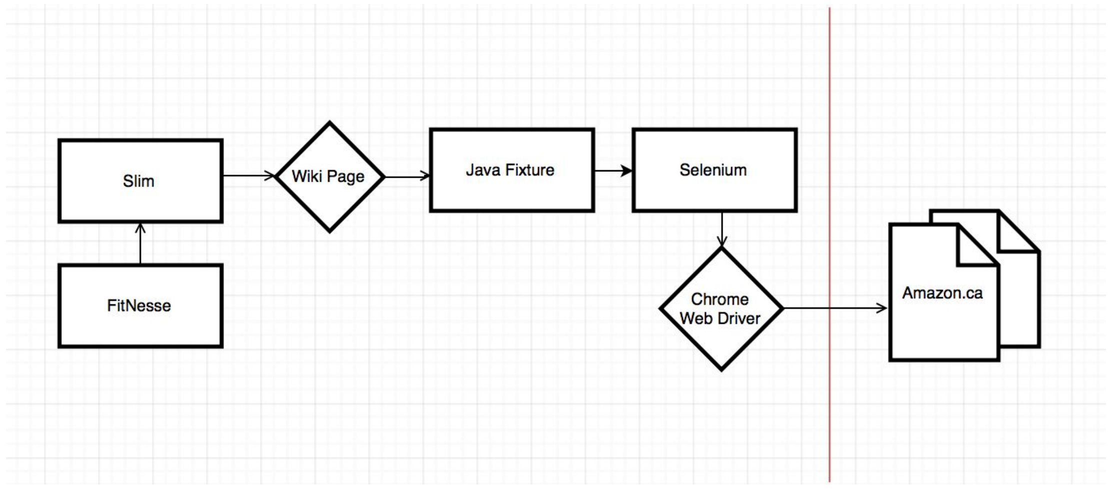
Cucumber

As shown in the diagram, Java encompasses all of the Cucumber Framework. With Java, we are creating the Gherkin scripts and the step definitions with Selenium to automate actions through Chrome.

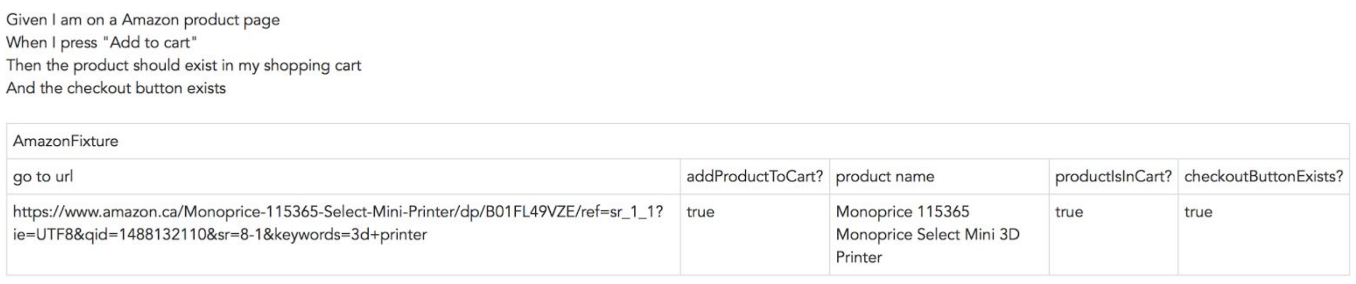


FitNesse

As shown in the diagram, FitNesse is a standalone application and uses built-in Slim testing system that reads a Wiki page and connects it to a Java fixture. The Java fixture then uses Selenium and the Chrome Web Driver to automate web tasks on Amazon.

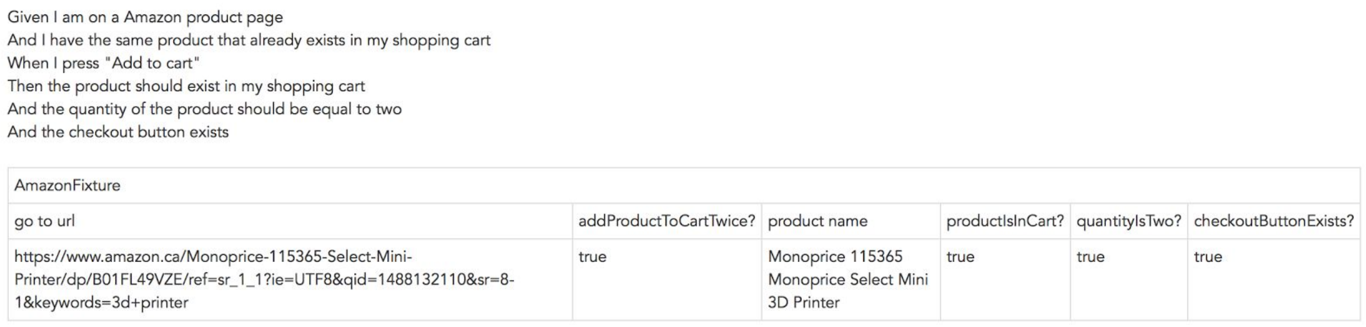


FitNesse Tables



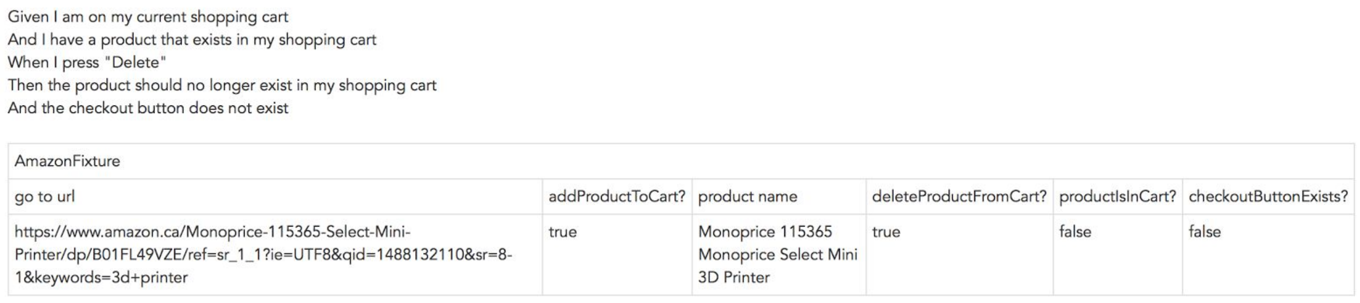
Given I am on a Amazon product page
When I press "Add to cart"
Then the product should exist in my shopping cart
And the checkout button exists

AmazonFixture				
go to url	addProductToCart?	product name	productIsInCart?	checkoutButtonExists?
https://www.amazon.ca/Monoprice-115365-Select-Mini-Printer/dp/B01FL49VZE/ref=sr_1_1?ie=UTF8&qid=1488132110&sr=8-1&keywords=3d+printer	true	Monoprice 115365 Monoprice Select Mini 3D Printer	true	true



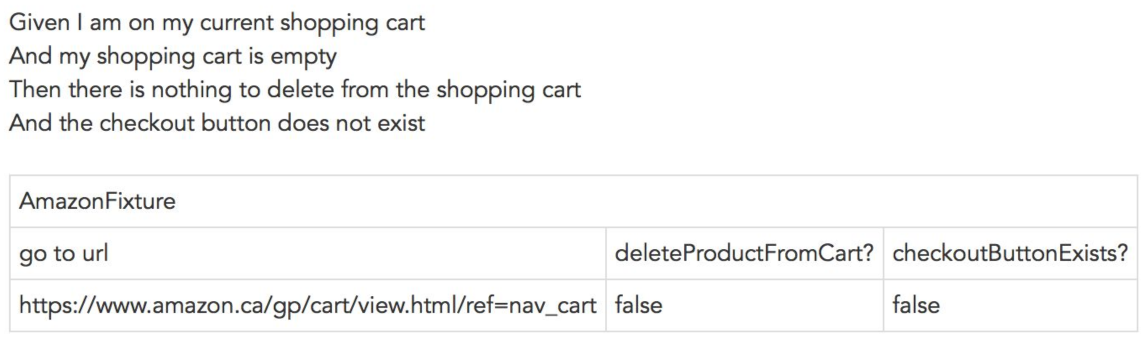
Given I am on a Amazon product page
And I have the same product that already exists in my shopping cart
When I press "Add to cart"
Then the product should exist in my shopping cart
And the quantity of the product should be equal to two
And the checkout button exists

AmazonFixture					
go to url	addProductToCartTwice?	product name	productIsInCart?	quantityIsTwo?	checkoutButtonExists?
https://www.amazon.ca/Monoprice-115365-Select-Mini-Printer/dp/B01FL49VZE/ref=sr_1_1?ie=UTF8&qid=1488132110&sr=8-1&keywords=3d+printer	true	Monoprice 115365 Monoprice Select Mini 3D Printer	true	true	true



Given I am on my current shopping cart
And I have a product that exists in my shopping cart
When I press "Delete"
Then the product should no longer exist in my shopping cart
And the checkout button does not exist

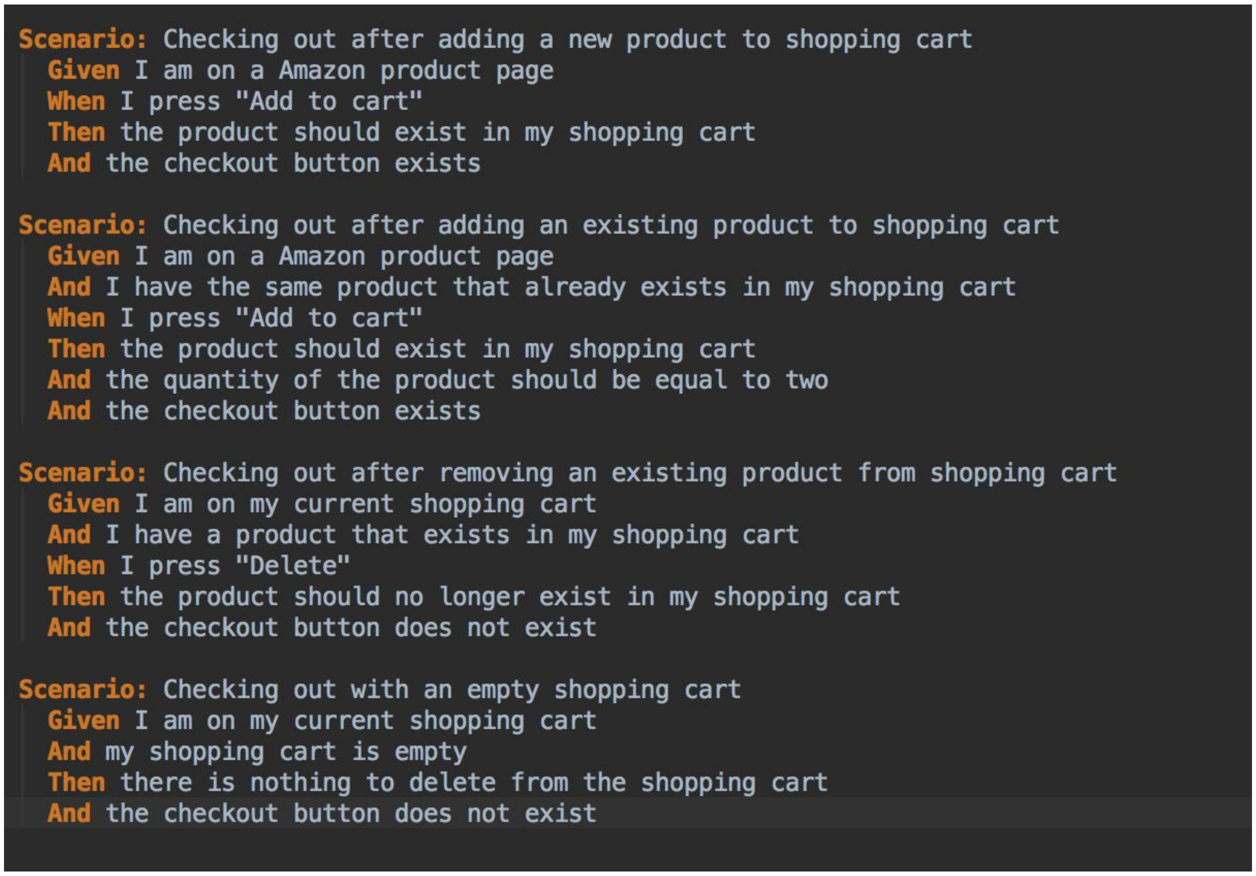
AmazonFixture					
go to url	addProductToCart?	product name	deleteProductFromCart?	productIsInCart?	checkoutButtonExists?
https://www.amazon.ca/Monoprice-115365-Select-Mini-Printer/dp/B01FL49VZE/ref=sr_1_1?ie=UTF8&qid=1488132110&sr=8-1&keywords=3d+printer	true	Monoprice 115365 Monoprice Select Mini 3D Printer	true	false	false



Given I am on my current shopping cart
And my shopping cart is empty
Then there is nothing to delete from the shopping cart
And the checkout button does not exist

AmazonFixture		
go to url	deleteProductFromCart?	checkoutButtonExists?
https://www.amazon.ca/gp/cart/view.html/ref=nav_cart	false	false

Cucumber Gherkin Scripts



```
Scenario: Checking out after adding a new product to shopping cart
  Given I am on a Amazon product page
  When I press "Add to cart"
  Then the product should exist in my shopping cart
  And the checkout button exists

Scenario: Checking out after adding an existing product to shopping cart
  Given I am on a Amazon product page
  And I have the same product that already exists in my shopping cart
  When I press "Add to cart"
  Then the product should exist in my shopping cart
  And the quantity of the product should be equal to two
  And the checkout button exists

Scenario: Checking out after removing an existing product from shopping cart
  Given I am on my current shopping cart
  And I have a product that exists in my shopping cart
  When I press "Delete"
  Then the product should no longer exist in my shopping cart
  And the checkout button does not exist

Scenario: Checking out with an empty shopping cart
  Given I am on my current shopping cart
  And my shopping cart is empty
  Then there is nothing to delete from the shopping cart
  And the checkout button does not exist
```

Source Code

Both Cucumber and FitNesse Java source codes can be found under one project:
https://github.com/kevin-nam/ecse428-assignmentb

For specifically Cucumber:

- Step definitions found under:
  - *src/test/java/com/kevinnam/cucumber/**StepDefinitions.java***
- Feature file found under:
  - *src/test/java/resources/**amazon.feature***

For specifically FitNesse:

- Fixture code found under:
  - *src/main/java/com/kevinnam/fitnesse/**AmazonFixture.java***
- Stand-alone FitNesse and root folder found under:
  - **fitnesse-standalone.jar**
  - **FitNesseRoot**/

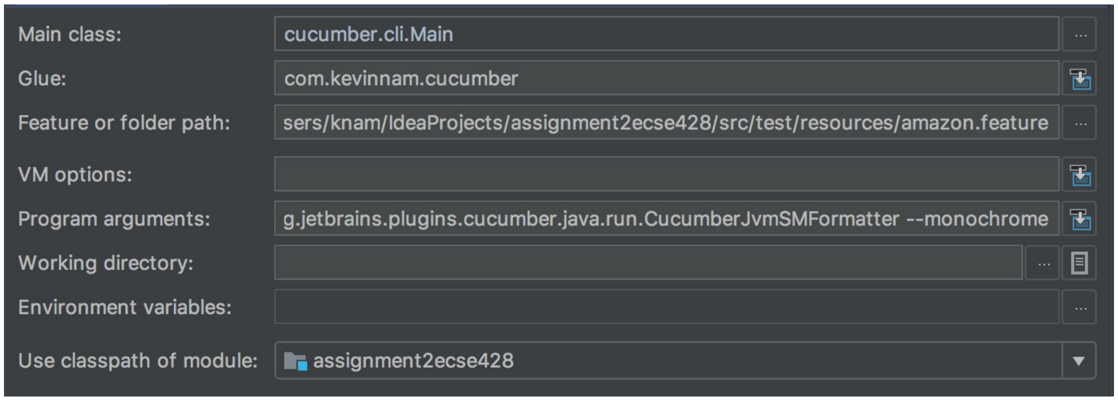
Instructions to install test environment and run tests (Cucumber):

First off, start by cloning the source code from the github repository using the link provided above (https://github.com/kevin-nam/ecse428-assignmentb).

Additionally, in order for Selenium to properly utilize the Chrome Web Driver, please download the driver here: https://sites.google.com/a/chromium.org/chromedriver/downloads

Using any IDE, edit *PATH_TO_CHROME_DRIVER* variable (line 21 in *StepDefinitions.java*) to be the absolute path to the newly downloaded Chrome Web Driver.

Using IntelliJ, build and compile the project and run *amazon.feature* with the following configurations:



Main class:	cucumber.cli.Main
Glue:	com.kevinnam.cucumber
Feature or folder path:	sers/knam/IdeaProjects/assignment2ecse428/src/test/resources/amazon.feature
VM options:	
Program arguments:	g.jetbrains.plugins.cucumber.java.run.CucumberJvmSMFormatter --monochrome
Working directory:	
Environment variables:	
Use classpath of module:	assignment2ecse428

Instructions to install test environment and run tests (FitNesse):

First off, start by cloning the source code from the github repository using the link provided above (https://github.com/kevin-nam/ecse428-assignmentb).

Additionally, in order for Selenium to properly utilize the Chrome Web Driver, please download the driver here: https://sites.google.com/a/chromium.org/chromedriver/downloads

Using any IDE, edit *PATH_TO_CHROME_DRIVER* variable (line 17 in *AmazonFixture.java*) to be the absolute path to the newly downloaded Chrome Web Driver.

Next, we need to download a stand-alone version of Selenium. You can find it here: http://www.seleniumhq.org/download/

Then, run a local instance of FitNesse on port 3000 by running the following commands in terminal: *java -jar fitnesse-standalone.jar -p 3000*

Go on any browser to localhost:3000 and navigate to the *Checking out a product Acceptance Tests* Wiki page. Click on Edit at the top and modify the classpath for selenium to the absolute path of your newly downloaded stand-alone selenium .jar. In addition, make sure the classpath to assignment2ecse428.jar file corresponds to that of your machine. If it doesn't exist, run a *mvn install* on the project.

Finally, back on the *Checking out a product Acceptance Tests* Wiki page, click on the Test button at the top of the page to run the tests.

Step by Step Test Process Flow (Cucumber)

Normal Flow:

1. Use Selenium to create an instance of Chrome Web Driver.
2. Use the driver to go to either a specified product page.
3. Click on **Add to cart button**. If button is not present, fail test immediately.
4. Use the driver to go to active cart page.
5. Check for the **presence of the product** in the shopping cart. If product is not present, fail test immediately.
6. Check for the presence of a **checkout button**. If checkout button not present, fail test immediately.
7. Close and quit the instance of Chrome Web Driver.

Alternate Flow:

1. Use Selenium to create an instance of Chrome Web Driver.
2. Use the driver to go to either a specified product page.
3. Click on **Add to cart button**. If button is not present, fail test immediately.
4. Re-do steps two and three.
5. Use the driver to go to active cart page.
6. Check for the **presence of the product** in the shopping cart. If product is not present, fail test immediately.

7. Check that the **quantity of the product** is equal to two. If quantity not present or quantity is not two, then fail test immediately.
8. Check for the presence of a **checkout button**. If checkout button not present, fail test immediately.
9. Close and quit the instance of Chrome Web Driver.

Error Flow:

1. Use Selenium to create an instance of Chrome Web Driver.
2. Use the driver to go to active cart page.
3. Click on **delete button** for products in the cart.
4. Check for the **presence of the product** in the shopping cart. If product is still present, fail test immediately.
5. Check for the presence of a **checkout button**. If checkout button is present, fail test immediately.
6. Close and quit the instance of Chrome Web Driver.

Step by Step Test Process Flow (FitNesse)

Normal Flow:

1. Use Selenium to create an instance of Chrome Web Driver.
2. Use the driver to go to either a product page as specified from FitNesse.
3. Click on **Add to cart button**. If button is not present, fail test immediately.
4. Use the driver to go to active cart page.
5. Check for the **presence of the product** in the shopping cart. If product is not present, fail test immediately.
6. Check for the presence of a **checkout button**. If checkout button not present, fail test immediately.
7. Close and quit the instance of Chrome Web Driver.

Alternate Flow:

1. Use Selenium to create an instance of Chrome Web Driver.
2. Use the driver to go to either a product page as specified from FitNesse.
3. Click on **Add to cart button**. If button is not present, fail test immediately.
4. Re-do steps two and three.
5. Use the driver to go to active cart page.
6. Check for the **presence of the product** in the shopping cart. If product is not present, fail test immediately.

7. Check that the **quantity of the product** is equal to two. If quantity not present or quantity is not two, then fail test immediately.
8. Check for the presence of a **checkout button**. If checkout button not present, fail test immediately.
9. Close and quit the instance of Chrome Web Driver.

Error Flow:

7. Use Selenium to create an instance of Chrome Web Driver.
8. Use the driver to go to active cart page.
9. Click on **delete button** for products in the cart.
10. Check for the **presence of the product** in the shopping cart. If product is still present, fail test immediately.
11. Check for the presence of a **checkout button**. If checkout button is present, fail test immediately.
12. Close and quit the instance of Chrome Web Driver.

FitNesse vs. Cucumber Implementation

Both FitNesse and Cucumber are well-established and useful testing frameworks. Each of these frameworks have their own individual strengths and weaknesses, and depending on the context and situation, one framework may be more beneficial than the other.

FitNesse is an incredible testing framework that promotes test-driven development. There is much use for FitNesse especially for non-technical users to create requirements and acceptance tests through the framework's unique markup styles driven with Wiki pages. As a technical developer, I found that FitNesse lacked in documentation when it came down to testing my Java code. It was difficult to understand why my Java code was not properly linked with the tables from a Wiki page. The ramp-up in learning FitNesse syntax handling is geared towards non-technical users but fails to facilitate developers in implementation. The error handling is not sufficient enough to detail, in the scope of FitNesse, the mistakes that were present in a Wiki page. But after solving certain problems, I can see the huge benefit in using this testing framework. It's incredibly flexible and allows for parallel collaborations (i.e. we can work on different requirements and tests at the same time in one place). In addition, the most powerful feature would be the fact that non-technical users can easily pick up on FitNesse and easily create requirements/tests before any code is written.

On the other hand, the Cucumber testing framework focuses on behaviour-driven development, creating acceptance tests and scenarios based on Gherkin scripts. Implementation can be done entirely on one IDE (e.g. IntelliJ) and run directly through a feature file. Using the Cucumber framework is less ideal for non-technical users who wish to create requirements and tests as it requires a degree of following syntax and the use of IDE. There is less flexibility in the sense that non-technical developers do not have as much freedom to create valuable documentation as with FitNesse. The upside with Cucumber is that implementation is much easier than FitNesse. Tests can be run directly from IntelliJ and there is no heavy ramp-up required when using Cucumber as compared to FitNesse because it is essentially just Java (or whatever language you are using). Other than the creation of Gherkin scripts (which is incredibly simple), there is no new “language” that you need to learn when using Cucumber.

All in all, FitNesse is better for non-technical users who wish to outline requirements and tests. Cucumber is better for technical users who don’t want the pain in learning new languages.

Video of Automated Tests

The videos of the automated tests have been uploaded on YouTube. I tried to get both the console logs and the automation of the web browser in the recordings.

Cucumber: https://youtu.be/PJpRHoba0pQ

FitNesse: https://youtu.be/YLCOUxo9p7o