October 31, 2017

Deliverable #3 Automated Testing Framework

1. EXPERIENCES

A. As per our constraints that were detailed in Deliverable 2, we have our limitations as far as meeting up in person outside of class. However, we did manage to meet up twice since the last deliverable. Our main issue moving forward are dependencies. Sigmah has thousands of dependencies throughout our code, which we are finding is creating issues when building specific java files. Imports and packages in scripts are new to us so we faced several issues with this. In short, we figured it out but in a cumbersome way using jar files and we chose to only test classes that had minimal dependencies for now. The way we understand dependencies is that it's sort of like importing something that you require, but that has already been done. So, on the surface, it seems as if much of the code that Sigmah requires was just repurposed from something else which begs the question if it is really necessary or if they are all just a series of shortcuts that were taken.

2. HOW-TO DOCUMENTATION <-- CHECK MAKE SURE THESE STEPS ARE ACCURATE AND RELATIVE

A. Requirements: Linux OS (Ubuntu 16.04.3 LTS), Java 1.8, Git

B. Clone GitHub Repository:
--> git clone https://github.com/CSCI-362-01-2017/NBA

C. Navigate to the TestAutomation file,
--> cd TestAutomation/scripts

D. Execute scripts/runAllTests.sh to run the framework and open the html file in the default browser.
--> ./runAllTests.sh

3. ARCHITECTURAL DESIGN

A. This testing framework is built using classes from Sigmah's source code. We are using a bash script, runAllScripts.sh, to create our automated framework. The script reads in all test case files, as they all follow the same format, then tests these test cases. Our test case files are stored as "testCase0#.txt" so that many more can be added; it leaves the framework open to additional tests. We have a Java Driver for each method that we are testing, which is compiled and ran with our script. The script is essentially the director of our testing framework, the brains if you will. The script compiles the driver, then runs the driver with the correct parameters which it reads in from the test case files. The output of the driver is then recorded in a report to show whether or not the test has passed or failed.

4. FILE STRUCTURE <--- NEEDS FINISHING

```
/TestAutomation
        /project
                /src
                        FaultedNumberUtils.java
                        NumberUtils.class
                        NumberUtils.java
                CleanNumberUtils.java
                FaultedNumberUtils.java
        /scripts
                ratioAsString_NumberUtilsDriver.java
                ratio_NumberUtilsDriver.java
                runAllTests.sh
                truncateDouble_NumberUtilsDriver.java
                truncate_NumberUtilsDriver.java
        /testCases
                Format.txt
                testCase01.txt
                testCase02.txt
                ...
                testCase24.txt
                testCase25.txt
        /testCasesExecutables
                ratioAsString_NumberUtilsDriver.class
                ratioAsString_NumberUtilsDriver.java
                ratio_NumberUtilsDriver.class
                ratio_NumberUtilsDriver.java
                truncateDouble_NumberUtilsDriver.class
                truncateDouble_NumberUtilsDriver.java
                truncate_NumberUtilsDriver.class
                truncate_NumberUtilsDriver.java
        /temp
                (for output from running tests ... be sure to clean at start
        of runAllTests)
                testCase1results (might be folder or file)
        /oracles
                testCase1Oracle (might be folder or file)
        /docs
                README.txt
                TeamProjectsSpecifications.pdf
        /reports
                NBA-TestReport.html
```

5. TEST CASES

A. The template for our test cases are as follows:

   i. <test case ID>
   ii. <description of the method's purpose>
   iii. <class file name *.java>
   iv. <method name with any parameters>
   v. <input>
   vi. <expected output>

B. Test Case Format Description

   i. The test case ID is just the number of the test case, starting at 00 and ending at 05 (for now) it leaves the potential for endless test cases to be added.
   ii. The description is what the method itself is actually doing.
   iii. The class file name is the name of the java class file in which the method is contained.
   iv. The method name is the name of the function within the class that is being tested. Parameters (input) should be listed here as well.
   v. Input is what the user enters when this method is called.
   vi. The expected output is what should be returned when this method is called with this specified input.