Software Requirement Tests

1. The solution shall use a minimum of Java 7 and a maximum of Java 8.

We tested this requirement by attempting to compile and build our project with Java 8, and saw that the build succeeded and that the project was fully functional.

2. The solution shall test executables in the form of Java Jars via the command line.

We made sure that the code accepted Java jars by testing several Java Jars on our software and checking whether or not the Jars had been read by printing out key attributes of the Jars.

3. The solution shall increase code coverage of the Software Under Test (SUT) via exploratory black box testing.

We saw that the percentage of code coverage increased after our additional tests were run.

4. The solution shall increase the number of unique exceptions generated by the SUT via exploratory black box testing.

We made sure that our tests actually exposed problems by manually looking at the crash data and ensuring that the application actually crashed on the given test.

5. The solution shall accept the path of the executable jar to test as a required command line argument in the form of: -jarToTestPath.

We tested this requirement by not only printing out the required path but also by printing out important attributes of the JAR file that was read to make sure that the correct JAR file was read.

6. The solution shall accept the path to the directory where jacoco will generate output files as a required command line argument in the form of: -jacocoOutputPath.

We tested this requirement by adding a breakpoint and printing out the path and then checking if the output files reached their correct addresses (for several different command line arguments).

7. The solution shall accept the path of the jacoco agent jar in the form of: -jacocoAgentJarPath.

We tested this requirement by adding a breakpoint and printing out the path to the jacoco agent jar.

8. The solution shall execute the specified number of exploratory black box test iterations (default 1000) against the SUT.

Each time a black box test was successfully run against the SUT, a counter was incremented and later it was checked whether the counter was equal to the specified number of exploratory black box test iterations.

9. The solution shall run additional exploratory black box test iterations, if the specified number of exploratory black box test iterations completes within a specified test time goal (default 5 minutes).

We made sure that this requirement was satisfied by running 10 quick tests and then checking whether or not additional black box test iterations were run within the specified time goal.

10. The solution shall accept the number of exploratory black box tests to run as an optional command line argument in the form of: -bbTests .

We tested this optional command line argument by creating a breakpoint and then printing out the given argument to check that the program read it in properly.

11. The solution shall accept the test time goal as an optional command line argument in the form of: -timeGoal .

We tested that the timeGoal commandline argument was accepted and read by adding a breakpoint in the code and then printing it out, and checking if it was right.

12. The solution shall support applications that take a fixed number of arguments.

We tested this requirement by trying applications with a fixed number of arguments and determining whether or not the arguments were correctly determined (through breakpoints and print statements).

13. The solution shall support applications that take a variable number of arguments.

We tested this requirement by trying applications with a variable number of arguments and determining whether or not the arguments were correctly determined (through breakpoints and print statements).

14. The solution shall support integer arguments.

We tested this requirement by passing in an integer argument, setting a breakpoint, and then utilizing a print statement to check whether or not the argument had made its way through correctly.

15. The solution shall support double arguments.

We tested this requirement by passing in a double argument, setting a breakpoint, and then utilizing a print statement to check whether or not the argument had made its way through correctly.

16. The solution shall support bounded String arguments.

We tested this requirement by passing in a bounded String argument, setting a breakpoint, and then utilizing a print statement to check whether or not the argument had made its way through correctly.

17. The solution shall support unbounded String arguments.

We tested this requirement by passing in an unbounded String argument, setting a breakpoint, and then utilizing a print statement to check whether or not the argument had made its way through correctly.

18. The solution shall record unique exceptions or crashes seen during exploratory testing.

We tested this functionality by first printing out all of the exception/crash data, determining the expected unique exception/crash data, and then comparing it to our printed unique exception/crash data.

19. The solution shall print to stdout a YAML report at the end of testing that is in the following format: Total predefined tests run: Number of predefined tests that passed: Number of predefined tests that failed: Total code coverage percentage: Unique error count: Errors seen: - - -

We tested this functionality by looking at the YAML report and comparing the format to the expected format.

20. The solution shall accept an optional command line argument of the form "-toolChain" that will limit the output to contain only the parseable YAML indicated above.

We tested this requirement by putting in the optional command line argument and checking whether the output was limited to contain only parseable YAML.

21. The solution shall be delivered as an executable jar with the name com.idtus.contest.winter2017.framework.jar.

We tested this requirement simply by looking at the project and checking whether or not the requirement was fulfilled.

22. The solution shall also be delivered as an Maven Project in Eclipse (zipped up) with the name com.idtus.contest.winter2017.framework.zip.

We tested this requirement simply by looking at the project and checking whether or not the requirement was fulfilled.