Project Report Part 1

CS362_400 F16

Due: 11/21/2016

Erin Sullens
Jason Ghiraldini
Keith Adkins
Markus Woltjer

Test Plan for Apache URLValidator

1.0 Current Test Coverage

This project is starting with no test coverage in place. We will be providing all needed test coverage.

2.0 Areas in which the tests could be improved using the tools and techniques described in class.

There are many tools covered in this class that our team could use to improve the testing of this project. To start, we would use simple static analysis to find any blatant errors in our code. This could also point out any errors in structure or order of operations in our tests. In junction with static testing, we will also utilize compiler options to display any warnings we have not caught ourselves. Another method for testing we will use is of course dynamic testing. For our dynamic testing we will create various test cases to build our suite. We will be doing white box testing in order to create more comprehensive tests by looking into the source code. With white box testing we will utilize mutation testing in order to verify our test suite is capturing all known bugs or mutants. One great visualization method we will be using will be various types coverage. Graph coverage and logic coverage will be used in various test cases. This will give us feedback on what we have tested and what we have not tested in the source. Input domain partitioning is another form of testing we will use in order to verify that different types of inputs produce the correct outputs. A mixture of all these tools will

create a very thorough and eclectic test suite for our project. The more comprehensive our test suite can be, the more likely we are to fix all the bugs in the source and deploy a great program.

3.0 A plan for implementing those improvements

For dynamic testing on the Apache URLValidator, we will create a suite of unit tests on the methods in UrlValidator.java, RegexValidator.java, InetAddressValidator.java, and DomainValidator.java. Using code coverage on the unit tests will determine if our test suite is complete. For input domain partitioning, making a random test for each partition would be effective. We could split up the partitions by the different schemes, and if fragments are present or not. We also plan to use mocking to test the URLValidator.java class so it can be tested without using any of the other classes.

4.0 At least one additional tool/method you plan on using not utilized in the assignments.

The additional tool we will be using is PIT, a mutation testing tool for use with Java in the Maven testing environment (http://pitest.org/). Basically, PIT provides a measure of how good our test suite is by randomly inserting bugs, called mutations, into the source code we are testing. If our test suite finds a mutation, PIT will kill it. Otherwise, the mutation will live. A living mutation is an indication of coverage that our test suite is missing. PIT will be a good addition to our test suite in helping us identify areas where we are missing test coverage.