Project Report Part I - Test Plan

Bradley Huffman, Caleigh Runge-Hottman, Jesse Hanson, Jonathan Nicolosi

Current Test Coverage

As it currently stands, running cobertura testing against the classes produces 0% code coverage as there are no test classes written or provided with this project.

Areas for Improvement

Considering the Buggy URL Validator does not have any test coverage to date, there are many areas for improvement. Our code coverage could be improved, and we plan to write tests for each of the classes in the URLValidator with a goal of achieving 80% coverage on all of the tests. Achieving 80% code coverage will ensure that we have executed the majority of the code base, which is important for finding bugs. We will also test each input type for the classes against correct and incorrect types and make sure they pass/fail accordingly. Moreover, we will report on any false-positives as well on ways we might resolve incorrect failures.

Test Plan

For the implementation of the improvements outlined above, we will be using JUnit as our testing framework. We will use Cobertura to gauge the amount of code coverage. To obtain the coverage, we will use random testing and functionality based input domain partitioning.

DomainValidator:
We will test various valid and invalid country codes, generic top-level domains, infrastructure top-level domains, and local top-level domains via functionality based input domain partitioning to ensure the valid codes and top-level domains return ‘true’, and the invalid codes and top-level domains return ‘false’.

InetAddress Validator:
We will confirm that the function correctly identifies IPv4 addresses and that it accurately verifies that address subgroups are legal.

UrlValidator:
For the url validator, we will ensure that we can both send and receive packets from the url as well as make sure the url resolves and is a navigable url. We will do so by checking the returns of the url validator and testing if they return true for proper urls and false for bad urls and what inputs on the url might cause it to pass or fail incorrectly

RegexValidator:
We will ensure that the RegexValidator function throws exceptions when appropriate. We will confirm that caseSensitive flag is being set properly.

ResultPair:
We will use pass fail tests to determine if the ResultPair function correctly matches the input to its respective part in the url. We will use these to find if there are input that incorrectly pass or fail and report on what we can do to prevent false positives.

Additional Methods
One additional method which our group will be utilizing for this project is PIT mutation testing. Mutation testing is a form of testing that involves incorporating mutations into one's source code, and then running the corresponding tests for said source code to see if they fail as a result of these mutations. The quality of the tests can then be determined by analyzing how many of the mutations cause test cases to fail, since ideally no mutations should survive the tests. We will run each of the classes against 100 mutation tests and report on the results. The report will reflect on the tests that had the most failed mutations and why this might have occurred.