Project Report Part I << Due: Mon, Nov 21st , 11:59p

Corinna Huffaker
Jeromie Clark
Bolan Peng
Damian Mecham

https://github.com/OSU-CS362-F16/f16-project-cs362-assignment-5-west-coast

Code Baseline

Project test plan is based on evaluating the BuggyUrlValidator code provided at http://web.engr.oregonstate.edu/~baldwdav/BuggyURLValidator.zip

Tests Based on Class Tools

There were no tests provided with the source code. The techniques covered in the class offer many areas for improvement. The following tests based on class tools are planned:

1) Unit tests on each of the classes: DomainValidator, InetAddressValidator, RegexValidator, ResultPair and UrlValidator.

   The goal with unit tests is maximum line coverage and maximum branch coverage. Many of the parameters of the classes are 'protected'. The technique of extending the class to provide access to these parameters will be used.

   If necessary, Mockito will be used to mock the inputs of classes to expand the coverage. Cobertura will be used to determine coverage and identify areas that are not covered by tests.

   Where possible boundary value tests will be included in the unit test input values.

   Input domain testing for various named segments of the URL will be used to break down the URL to enhance unit test coverage.

2) A randomizer will be constructed to test different combinations of allowed domain, TLDs, options, strings and upper/lower case letters. Java provides a URL validator that may be used as an Oracle to test the validity of the randomly generated URLs.

http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/httpThreadLink/HttpClient.html#section3

3) Several lists of known test cases have been identified for use in testing the validator.

   The goal will be to read in the url test cases from an input file along with an expected result.

   Example sources of known test cases:
   http://greenbytes.de/tech/webdav/urldecomp.xml
   https://mathiasbynens.be/demo/url-regex

Tests Based on Additional Tools

The plan will be to attempt using the PIT tool for Mutation testing of the unit tests developed in part 1. The PIT tool is available at: Mutation testing The objective of mutation testing is to evaluate the quality of the test code by injecting faults into the classes. If the test fails, the mutation is killed. Tests that don't fail may indicate inadequate tests.