CS362 Final Project Part B
December 6, 2015

Project Members

Larissa Hahn
Solomon Huynh
Dustin Chase

Introduction

This document will explain our testing of the URLValidator. It will go through our methodology for testing, our testing results, and how we worked as a team.

Teamwork

Using Skype, Google Hangouts, and Email, we worked together well as a team over Skype mostly and worked at the same time which definitely helped us to collaborate and help each other with all the tests that we wrote. We went through all the URLValidator code together, making sure the understanding of what needed to be tested was there, and then the work was divided up fairly evenly among all team members, in a pair programming scenario, where each member was available and chimed in to help the others as we wrote all the tests together, with one teammate assigned a certain portion of the test to write, and so on. All three of us wrote the testIsValid() test together. All of us wrote in the report on the bugs that each of us found as well as the tests that we wrote that discovered these bugs. Overall it was great team effort.

Methodology of Testing

Our first approach was to experiment with some manually generated URLs which just contained IP addresses. We generated the following list of invalid IP addresses:

String[] invalidAuth = new String[]{"256.256.256.256", ".1.2.3.4", "go.a", "1.2.3", "123.123.456.123", "123.123.0.456", "1.2.3.4.", "1.444.1.1", "1.2.3.4.5", "0.999.0.999"};

Our strategy was to use different ranges of numbers to see if the validator would pick up those that were out of range for valid IP addresses. The incorrect numbers were assigned to different places in the IP address so that, if any errors were found, we could isolate where in the sequence the validator was not finding the error.

To our surprise, these simple tests picked up an error right away. The validator erroneously found many of these invalid IP addresses to be valid. The resulting output showed these results:

Error: 256.256.256.256 is valid when it should be invalid.
Error: 123.123.456.123 is valid when it should be invalid.
Error: 123.123.0.456 is valid when it should be invalid.
Error: 1.444.1.1 is valid when it should be invalid.
Error: 0.999.0.999 is valid when it should be invalid.

The total number of errors found is 5.

Examining these results, we found that it didn't matter where the incorrect numbers were in the URL. It appeared that it was just the range of numbers was being validated incorrectly. We noted this preliminary test result and moved on to the next set of tests.

Partitioning

We partitioned the URLs by the different parts such as IP address and port number. Our strategy was to uncover which parts of the URL would be correctly validated, and then look at different combinations of those parts at the end of the testing to make sure that if the validator correctly evaluated a part of the URL, that it would not fail when the different parts were combined.

One aspect of the URLs we partitioned was the top level domain. At first, it didn't seem to the group that this would be a good place to look for bugs since it was so obvious that the validator should correctly validate this aspect of the URL. After running a few manual tests in this partition, we quickly found that we had to investigate this further. We found an issue with the country domains not being picked up correctly. The test output was as follows:

http://www.google.kr, is valid.
http://www.google.lr, is valid.
http://www.google.mx, is valid.
http://www.google.nz, is valid.
http://www.google.om, is valid.
http://www.google.pl, is valid.
http://www.google.qa, is valid.
http://www.google.ru, is valid.
http://www.google.se, is valid.
http://www.google.th, is valid.
http://www.google.uk, is valid.
http://www.google.vn, is valid.
http://www.google.wf, is valid.
http://www.google.ye, is valid.
http://www.google.zw, is valid.

The validator was telling us that these URLs were invalid when they were actually valid. Examining these more closely, we noticed that these were all country extensions. Then we found that the country codes with errors only occurred after the letter 'j'. We filed away this observation and proceeded with the rest of the testing.

Tests Executed

testManualTest()

- Manually test valid web links such as "http://www.google.com/maps".
- Manually test invalid web links to see if it caught the error.
- Manually test valid authorities.
- Manually test invalid authorities to see if it caught the error.

testYourFirstPartition()

- Store an array of valid country top level domains with only one country of every letter of the alphabet. Then run the isValid function to confirm if they are return true for being a valid domain.
- Store an array of random non-existent top level domains and then run isValid function to confirm that the function returns false for invalid domain.

testYourSecondPartition()

- Randomly generate port numbers from the range of -10 to 70000 to and test it with a valid IP address with the isValidAuthority function to see if the function returns true for being a valid port number and false for invalid port number.

testIsValid()

- Build an array of strings for valid/invalid schemes, authorities, paths, IP address, and port numbers.
- Build combinations schemes, authorities, and paths and test them with the isValid function to check if the combination is correct or incorrect like it should be.
- Build combinations of the IP address and port numbers and test them with the isValidAuthority function to check if the combination is correct or incorrect like it should be.

Agan's Principles

David J. Agan has nine principles in software testing that we were able to make use of in debugging URLValidator. For "Understanding the System", our team got together over Skype and went through the URLValidator code line by line. In this way it would allow us to understand the fundamentals and different parts of URLValidator and what needed to be tested. We accomplished this as Part a of the Final Project. We also did the "Make it fail" as well in order to understand the limits of URLValidator and know what valid input or invalid input would be for our partitioning. So we guessed then only to focus the search for our tests in finding bugs. We also definitely used #5 "Change one thing at a time" to isolate what was causing the failures in order to find the source of the bugs. Some of them we did not make use of simply because we are not fixing the bugs merely finding and reporting on them.

Bug Reports

Bug Report 1
Title: Port Numbers Incorrectly Labelled as Invalid when They Have More than 3 Digits
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URLValidator
Version: 1.4 - Revision: 1227719
Components: Validation Routine
Classification: A
Can this bug be reproduced (how often?): Yes, this bug can be reproduced. Every time a port exceeding 3 digits is submitted to isValid(), the URL is incorrectly labelled as invalid.

Description:

Summary: Input tested "http://amazon.com:2500"
What Happened: Method returned false for a valid input
Supporting Information: JUnit test method "testIsValid"
In what lines and in what file did the failure manifest itself in: URLValidator.java: Line 158, regular expression limits port number to maximum of 3 digits
Steps to Reproduce Bug: Run JUnit tests, or test individual URI with port number of more than 3 digits
Expected Results: Returns "true"
Actual Results: Returns "false"
Workaround: N/A
Effect/Implications of Bug: Attempting to connect to valid port numbers are stated to be invalid. Ports with more than 3 digits are not allowed when they should be.

Bug Report 2
Title: Country Code TLDs Missing
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URLValidator
Version: 1.4 - Revision: 1227719
Components: Validation Routine
Classification: A
Can this bug be reproduced (how often?): Yes, each time.

Description:

Summary: Input tested: top level domains "kr", "lr", "mx", "nz", "om", "pl", "qa", "ru", "se", "th", "uk", "vn", "wf", "ye", "zw".
What Happened: The URL validator indicated these were invalid when they are actually valid country codes.
Supporting Information: Run JUnit tests
In what lines and in what file did the failure manifest itself in: DomainValidator.java: Lines 249 to 357. Missing all countries past those that start with the letter 'i'.
Steps to Reproduce Bug: Run JUnit tests
Expected Results: Returns "true"
Actual Results: Returns "false"
Workaround: N/A
Effect/Implications of Bug: Cannot validate correct top level domains

Bug Report 3
Title: IPv4 Outside Correct Range Returns Valid
Type: Bug
Status: Open
Priority: A
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URLValidator
Version: 1.4 - Revision: 1227719
Components: Validation Routine
Classification: A
Can this bug be reproduced (how often?): Yes, each time the tests are run

Description:

Summary: Input tested "256.256.498.256"
What Happened: Method returned true for an invalid input
Supporting Information: JUnit test method "testIsValid"
In what lines and in what file did the failure manifest itself in:
Steps to Reproduce Bug: Run JUnit tests, or test individual URI as noted above
Expected Results: Returns "false"
Actual Results: Returns "true"
Workaround: N/A
Effect/Implications of Bug: Upper bound on IP addresses is not being correctly validated.