CS 362 - Final Project Part B

Jason Gourley
James Riccelli
Michael Walker

Testing Methodology

We applied the testing methodologies that we have been utilizing through the course to research and identify test cases to show the effectiveness of the program to meet the requirements for the software under test.

The first step for all testing was to research the structure of valid URIs. In order to verify the software under test's validity we needed a good understanding of how to construct test cases which would prove to be reliable for finding bugs. We first sought out the RFCs which establish the rules for URIs, IPv4, and IPv6 addresses. Looking these documents over gave us rules to follow when constructing test cases and how to identify edge cases. Secondly we took a look at other elements which are governed by organisations such as the IANA. These would include elements such as URI schemes as well as top level domain names. Reviewing these lists gave us practical cases which can be verified beyond just structure data in the RFCs. Utilizing these lists we could begin targeting specific areas that could present issues, such as valid TLDs which do not appear in the array used by the software to verify if it is valid or not.

With basic structure knowledge along with ideas to move forward on specific test cases we began to break down test cases into the three outlined areas. First manual tests, where we could test some of the basic edge cases and provide a set of tests that were straightforward and might catch errors in more widely used URIs. Second we created a set of partition tests to isolate the URI scheme and then the combination of host names and TLDs. Last we created a set of unit tests to close out our test suite. ***All of our code used for testing can be found at URLValidator/src/UrlValidatorTest.java.***

Manual Testing: ***testManualTest()***

For manual testing, it was important to find URL's that were valid and not valid, by accruing many different, functional, URL's with diverse TLD's. The URL's that were manually were each run through the isValid() method. The test would simply print out our expected value and the actual value returned by the function. These manual tests were run in junit and reported some bugs which will be discussed below in their associated bug reports. ***All URLs used for manual testing can be found in our group's UrlValidatorTest.java file under the method testManualTest().***

Partitioning: *testYourFirstPartition() & testYourSecondPartition()*

For partitioning it was important for our test suite that we looked at the basic necessary building blocks of a URI to verify the software under test prior to moving forward with any other unit tests. After reviewing the corresponding RFCs we identified the two most important areas to do partitioning upon was the scheme and then the authority, combination of hostname and TLD. These are the two most important parts of URI for testing as the other elements of a URI are dependent upon reliable testing of these parts. A failure in either of these two areas would cause any other tests of the other structural elements to be brought into question.

In the first partition test, URI schemes, it was important to understand the structure, but also the URIs that are approved. A list of tests were created based upon first the RFC reference testing the boundaries for cases involved with the makeup of a valid scheme. In RFC 3986 this is listed as being : scheme = ALPHA *(ALPHA / DIGIT / "+" / "-" / "."). A set of tests was built around these requirements after brainstorming ideas on what edge cases could exist from this structure. The next step was to search out the list of approved URI schemes that are live and need to be verified. For the purposes of this round of testing the included schemes were limited to schemes that have been moved to a permanent status.

For the second set of partition tests regarding the authority structure we needed to take special care to look into valid host names, label structure, TLD structure, approved TLDs, IPv4 and IPv6 addresses. There are a number of elements that are combined at this level which require care to understand the structure and validity of the tests created. In creating the test cases we approached individual elements as possible to test edge cases and basic structure. The first tests that made sense were to ensure that IP addresses were being validated correctly since they are not connected in the way host names and TLDs are. Next to cover the hostnames the basic structure rules were taken into account for the valid lengths, characters, format. Edge cases for the hostnames were created using consistent TLDs where an error at the TLD level could be easily identified. For the TLDs the basic structure was first taken into account and then various basic domains were selected as a basic starting point.

Unit Tests: *testIsValid()*

Our unit test took a similar approach to that of the testIsValid method found in the UrlValidatorTest.java file in the URLValidatorCorrect directory when it came to testing valid and invalid URLs. We split URLs out into its individual segments which comprise the URL as a whole. URL segments, as split by our group, consisted of scheme, subdomain, domain, port, and path. To establish a base collection of URL subgroups we established the minimum value of all URL segments, which basically equated to a two part question: 1) can the URL segment be an empty string and 2) if not, what is the minimum number of elements could a URL segment consist of to be considered valid. It turns out that every segment of the URL could be an empty string except the domain itself which consisted of a domain name and its TLD, such as “google.com”. Once we had our base cases established we could move on to adding special elements and/or cases to each segment group. In addition to empty strings tested on each segment, each segment had a space inserted (in segments that were not empty strings) and checked for validation since a URL does not allow the use of any spaces.

The scheme segments tested various protocols, such as http, https, imap, etc. Various combinations of colons and forward slashes were tested in addition to the one acceptable combination of “://”. The subdomain was tested by using various subdomain such as “video.”. The domain segment was tested with back and forward slashes (both of which being invalid), as well as various TLDs such as “.com” and “.net”. The port segment was tested by checking various port numbers starting at zero on up. A major bug in the code was found by using this strategy, in that we discovered that ports exceeding three digits were always deemed invalid. This is incorrect as computer ports can go up to values of 65535. Finally, the path segment was tested by using various combinations of characters with a focus on the forward slash character, such as double backslashes which are not valid, as well as invalid characters including question marks. All in all, our testing strategy was successful in that we found the bug which incorrectly limited port numbers to numerical values of three digits or less.

Working As A Team

For team collaboration we chose to communicate through email, google hangouts, and google drive as necessary to plan for the individual project parts, allowing for the flexibility needed to match all team members schedules. Together as a team we looked at the project requirements for the deliverables for both part A and B. Then we broke each major requirement down into atomic parts, which were grouped together by similarly logical parts allowing each team member to focus on an individual area of the software under test.

Bug Report 1

Title: IPv4 Address Out of Valid range Returns True
Type: Bug
Status: Open
Priority: Blocker
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URL Validator
Version: 1.4 – Revision: 1227719
Component(s): Validation Routines
Classification (How Serious is the Bug): A
Can the Bug be Reproduced (If So, How Often): Yes, everytime
Description

- ***Summary:*** Input tested “http://256.256.256.256”
- ***What Happened:*** Returned invalid results for input
- ***What’s the Error Code/Message (If Applicable):*** N/A
- ***Supporting Information (if Applicable)***: JUnit test method “testYourSecondPartition()”

In What Line(s) and In What File did the Failure Manifest Itself In:
InetAddressValidator.java : Line 96, IP address segment greater than 255 returns true, however should return false
Steps to Reproduce Bug: Run JUnit tests, or test individual URI as input above
Expected Results: Returns “false”
Actual Results: Returns “true”
Workaround (If Applicable): N/A
Effect/Implications of Bug: IP addresses are not correctly being checked for upper bounds
Attachments: RFC 1918, IPv4 Specifications
Contact Information:

- Jason Gourley – jpgourley@gmail.com
- James Riccelli – jrriccelli@gmail.com
- Michael Walker – mwalker.uw@gmail.com

Bug Report 2

Title: Hostname Exceeding Valid Range is Validated
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URL Validator
Version: 1.4 – Revision: 1227719
Component(s): Validation Routines
Classification (How Serious is the Bug): B
Can the Bug be Reproduced (If So, How Often): Yes, everytime
Description

- ***Summary:*** Input tested "http://abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcde1abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijk2abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijk3abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuv.com"
- ***What Happened:*** Returned invalid results for input
- ***What's the Error Code/Message (If Applicable):*** N/A
- ***Supporting Information (if Applicable)***: JUnit test method "testYourSecondPartition()"

In What Line(s) and In What File did the Failure Manifest Itself In:
DomainValidator.java : Line 136 , group length is tested for a minimum length but not a maximum length
Steps to Reproduce Bug: Run JUnit tests, or test individual URI as input above
Expected Results: Returns "false"
Actual Results: Returns "true"
Workaround (If Applicable): N/A
Effect/Implications of Bug: URIs can be validated that exceed the specifications for URI length
Attachments: RFC 3986, Section 3.2.2 Host
Contact Information:

- Jason Gourley – jpgourley@gmail.com
- James Riccelli – jrriccelli@gmail.com
- Michael Walker – mwalker.uw@gmail.com

Bug Report 3

Title: Unable to Validate Certain Country Code TLDs
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URL Validator
Version: 1.4 – Revision: 1227719
Component(s): Validation Routines
Classification (How Serious is the Bug): A
Can the Bug be Reproduced (If So, How Often): Yes, everytime
Description

- ***Summary:*** Input tested : "http://1abc.us", "http://abc1.us", "http://abc.us", “http://test.us”, “http://www.watchfree.to/”
- ***What Happened:*** Returned invalid results for input
- ***What’s the Error Code/Message (If Applicable):*** N/A
- ***Supporting Information (if Applicable)***: JUnit test method “testYourSecondPartition()”

In What Line(s) and In What File did the Failure Manifest Itself In:
DomainValidator.java : Line 248 COUNTRY_CODE_TLDS, missing “us” for United States from the country code array
Steps to Reproduce Bug: Run JUnit tests, or test individual URIs as input above
Expected Results: Returns “true”
Actual Results: Returns “false”
Workaround (If Applicable): N/A
Effect/Implications of Bug: Unable to validate all correct top level domains
Attachments: http://data.iana.org/TLD/tlds-alpha-by-domain.txt, Latest top level domains provided by IANA
Contact Information:

- Jason Gourley – jpgourley@gmail.com
- James Riccelli – jrriccelli@gmail.com
- Michael Walker – mwalker.uw@gmail.com

Bug Report 4

Title: Protocol Extension Error
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URL Validator
Version: 1.4 – Revision: 1227719
Component(s): Validation Routines
Classification (How Serious is the Bug): A
Can the Bug be Reproduced (If So, How Often): Yes, every time junit is run
Description

- ***Summary:*** Input tested "h:///www.craigslist.org/"
- ***What Happened:*** Method returned true for valid url
- ***What's the Error Code/Message (If Applicable):*** N/A
- ***Supporting Information (if Applicable)***: attached

In What Line(s) and In What File did the Failure Manifest Itself In: urlValidator.java : line131
Steps to Reproduce Bug: Run JUnit tests
Expected Results: False
Actual Results: True
Workaround (If Applicable): N/A
Effect/Implications of Bug: URLs with incorrect protocols are deemed valid by isValid() method
Attachments: N/A
Contact Information:

- Jason Gourley – jpgourley@gmail.com
- James Riccelli – jrriccelli@gmail.com
- Michael Walker – mwalker.uw@gmail.com

Bug Report 5

Title: Port Numbers Exceeding Three Digits are Incorrectly Labeled Invalid
Type: Bug
Status: Open
Priority: Major
Resolution: N/A
Fix Version: N/A
Product: Apache Commons URL Validator
Version: 1.4 – Revision: 1227719
Component(s): Validation Routines
Classification (How Serious is the Bug): A
Can the Bug be Reproduced (If So, How Often): Yes, everytime a port exceeding 3 digits (ex: 1000, 23456, etc) is inputted into isValid(), the URL is incorrectly labeled invalid
Description

- ***Summary:*** Input tested "http://google.com:1000"
- ***What Happened:*** Method returned false for a valid input
- ***What's the Error Code/Message (If Applicable):*** N/A
- ***Supporting Information (if Applicable)***: JUnit test method "testIsValid()"

In What Line(s) and In What File did the Failure Manifest Itself In: UrlValidator.java : Line 158 , regex limits port number to max of 3 digits
Steps to Reproduce Bug: Run JUnit tests, or test individual URI as input above
Expected Results: Returns "true"
Actual Results: Returns "false"
Workaround (If Applicable): N/A
Effect/Implications of Bug: URLs attempting to connect to valid port numbers are deemed invalid, limiting ports used from 0-999.
Attachments: N/A
Contact Information:

- Jason Gourley – jpgourley@gmail.com
- James Riccelli – jrriccelli@gmail.com
- Michael Walker – mwalker.uw@gmail.com

EXTRA CREDIT:

Our group created a random tester for the isValid() method. The random tester creates URL segments, as described in the section above titled: “Unit Tests: testIsValid(),” each (other than the port segment) of a random length between two variables titles minString and maxString (set to 1 and 20 respectively at the time of testing) consisting of characters [a-z]. Each segment has a total of 10 unique strings created for the complete URL string, in total generating 100,000 unique URLs consisting of randomly generated strings. *(Note: The port segment was artificially capped at 999 due to a bug in the code improperly rejecting any URL with a port with more than 3 digits.)*

The code for the random tester can be found at URLValidator/src/UrlValidatorTest.java and the method is called testRandom().