Luke Brewbaker
John Brown
Gunnar Martin
CS362 - Spring 2016

**Final Project Report - Part B**

**Methodologies of Testing**

Manual Testing

One of the tasks we were asked to complete was manual testing. This testing was the most basic type of testing that we completed. In our approach to manual testing, we asked the most the fundamental questions of testing: What inputs should produce a “good” test? What inputs should produce a “bad” test?

To determine the “good” tests, we picked a number of URLs that we know should be valid. Here are a few of the inputs that we tried:

- http://www.google.com
- https://stuff.amazon.com/main
- ftp://oregonstate.edu/
- http://myWebSite.net/login?id=1234
- http://nike.com/shoes:1300

To determine the “bad” tests, we picked a number of URLs that we know should not be valid. Here are a few of the inputs that we tried:

- http:///www.google.com
- httpz://amazon.net
- ftp://oregonstate.skoolz
- http://mySite.net/login?theQuery////isBad()

After picking a plethora of good and bad URLs that had a mix of different URL components that were valid and invalid, we felt that we had given manual testing a good amount of coverage.

Input Partitioning

Each input is divided based on the different URL components: Scheme, Authoirty, Path, Fragment, and Query. Each component of the input is then divided even further based on specific properties. Both valid and invalid partitions are tested, and each partition is tested at least once. The test functions located in UrlValidatorTest.java are: testScheme(), testPath(), testQuery(), testAuthorirty(), and testFragment(). Below are a few examples of valid and invalid URL’s.

Scheme:
Valid: http://www.google.com
Invalid: ht://www.google.com

Authority:
Valid: http://www.gooogle.com
Invalid: http://www.!@@&^%$@$$%google.com

Option:
Valid: http://www.google.com/googllywoogly
Invalid: http://www.google.comgooglywoogly

Queries:
Valid: http://www.google.com?query=true&otherquery=false
Invalid: http://www.google.com?query==&

Port:
Valid: http://www.google.com:88
Invalid: http://www.google.com:aport

Unit Testing

To accomplish the unit testing we wrote a short Junit test to automatically run through an array of good and bad URLs and assertion check to see if the correct truth value was returned from the isValid() method. This tester was not as sophisticated as what was used with the correctURLValidator as we supplied strings containing good and bad URLs rather than having the program loop through all of the permutations. The test prints each URL it is testing to screen. This way we were able to tell which input produced an assertion fail. The bug found and the the efforts in localization and determining the cause are detailed in my bug report: VALIDATOR-1003

Test Code

```
  public void testAnyOtherUnitTest()
  {
          UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);

          String[] goodTests = new String[NUM_TESTS];
          String[] badTests = new String[NUM_TESTS];

          getGoodTests(goodTests);
          getBadTests(badTests);

          //good tests
          for(int i = 0; i < NUM_TESTS; i ++ ){
                  System.out.println("testing: " + goodTests[i] + "\n");
          assertTrue(urlVal.isValid(goodTests[i]));
          }
          //bad tests
          for(int i = 0; i < NUM_TESTS; i ++ ){
                  System.out.println("testing: " + badTests[i] + "\n");
          assertFalse(urlVal.isValid(badTests[i]));
          }
  }
```

**URLs Tested**

VALID:

"http://www.google.com";
"http://www2.yahoo.com/test/me";
"http://m.amazon.com?test=this";
"http://www.google.com:80";
"http://255.255.255.255";
"https://0.0.0.0";
"https://www.yahoo.com:50000"; Failed
“http://255.255.255.255?test=this&test2=that";
"https://0.0.0.0/test/me";
“https://www.amazon.com:0/go";

Invalid

"https:///www.google.com";
"https:///aserefadasd";
"https:///";
"https:///go.com/anything/testme";
"https:///bad.test";
"https:///www.google.com";
"https://a.";
"https:-65499";
"https://.a.*";
"https://1.2.3.4.5";

**Division of Work and Collaboration**

We knew from the beginning of the project that our ability to effectively split up the work would maximize our effectiveness as testers. We started by first carefully reviewing the requirements of the project and breaking them down into similarly sized tasked. We identified the three primary task as the following: First, we would need to perform manual testing on the function by simply plugging in different URLs that we thought would be valid or invalid. Second, we would need to partition the inputs and then plug in inputs from all the different partitions into the function. Third, we would need to write code for an automated tester to generate test cases.

After we broke the requirements down into three tasks, we each chose a task complete. We agreed that as soon as a team member completed his task, that he would write the bug report and share his findings with the team members. Using this approach, we would be sure not to write up separate bug reports on the same bug.

Our primary mode of collaboration was Google Hangouts, which allowed us to have a running dialog of how each team member was progressing. This approach to collaboration allowed us to alway be looped into each other's progress and gave us the ability to quickly respond if there were any questions or issues. For writing our bug reports, we created a document on Google Drive, so that we would all have real-time access to the reports.

In the end, we we all able to successfully research and explain our separate methods of testing the URLValidator code. We each found a bug to write a bug report for and got to work explaining each bug, and the process we used to localize it.

**Agan's Rules**

Before we started debugging, we gave a good review of David Agan's rules for debugging, to make sure we were applying these valuable principles as we set about our testing and debugging.

#1: Understand the System
#2: Make It Fail
#3: Quick Thinking and Look
#4: Divide and Conquer
#5: Change One Thing at a Time
#6: Keep an Audit Trail
#7: Check the Plug
#8: Get a Fresh View
#9: If You Didn't Fix It, It Ain't Fixed

**Bug Reports**

Bug ID: VALIDATOR-1001
Title: URL Validator isValid method returns FALSE, when given a valid URL with a legitimate query string

-- Details --
Type: Bug
Priority: Major
Environment: Eclipse IDE Running on Windows 10 64-bit
Status: Open
Resolution: To Be Fixed
Assignee: Gunnar Martin
Reporter: Gunnar Martin

-- Description --
Failure: URL Validator isValid method returns FALSE, when given a valid URL with a legitimate query string. For example the query string: "http://myWebSite.net/login?id=1234” returns FALSE, even though it is a valid URL.

Discovery: This failure was found during the running of a sequence of manual tests. The URL string "http://myWebSite.net/login?id=1234” was passed into a testing function, along with the expected result, TRUE. The actual result, FALSE, did not match the expected result, therefore the testing function returned FALSE, indicating that the test had failed. After testing the URL string "http://myWebSite.net/login” and getting a TRUE result, it was determined that the query string was causing the failure.

Cause: After discovery of the failure, the test with the URL query string "http://myWebSite.net/login?id=1234” was run through the debugger within the Eclipse IDE. By stepping through the code and watching carefully for the exact point where isValid() returns FALSE , the cause of the bug was localized to line 446 in the file “UrlValidator.java” within the function isValidQuery(). This function carries out the inspection of the query component of each URL. It is supposed to check whether the query component matches the specification for a valid query and return TRUE is it is valid, or FALSE if it is not. Since the return value had an erroneously placed NOT operator in front of the function returning a value for the isValidQuery() function, the code was erroneously returning FALSE, when it should be TRUE. When this line wass re-written the test function began running successfully.

Original line of code:
`return !QUERY_PATTERN.matcher(query).matches();`

Rewritten line of code:
`return QUERY_PATTERN.matcher(query).matches();`

We did utilize Agan’s Rules of Debugging to find this bug. The most important was probably rule 3, to look at the program as it is running, instead of just staring at the code and thinking about what might be wrong. The cause of the bug only revealed itself when we actually watched the test run, line-by-line and saw exactly where the test was coming back FALSE.

Bug ID: VALIDATOR-1002
Title: URL Validator isValid method returns FALSE, when given a valid URL with a localhost domain

Type: Bug
Priority: Major
Environment: Eclipse IDE Running on Windows 8.1 64-bit
Status: Open
Resolution: To Be Fixed
Assignee: Luke Brewbaker
Reporter: Luke Brewbaker

-- Description --
Failure: Local domain failure in DomainValidator.Java. Local host should be a valid input for URLValidator, but it fails. Localhost is a local domain type and accord to its function description it should be true.

Discovery: Any URL with localhost failed the test. The bug was localized by stepping through the code using the debugger in the Eclipse IDE.

Cause: The function is checking a negation of the value and returning true. It should be checking the value and returning true.

Original code: In DomainValidator.Java, Line 139 to 141:

```
if (!hostnameRegex.isValid(domain)) {
  return true;
}
```

Re-written code: Should be:

```
if (hostnameRegex.isValid(domain)) {
  return true;
}
```

Removing the “!” at the beginning should fix this issue.

Bug ID: VALIDATOR-1003
Title: URL Validator isValid method returns FALSE, when given a valid URL with a port number greater than 3 digits

-- Details --
Type: Bug
Priority: Major
Environment: Eclipse IDE Running on Windows 10 64-bit
Status: Open
Resolution: To Be Fixed
Assignee: John Brown
Reporter: John Brown

-- Description --
Fails on good input whenever testing a URL with a port greater than 3 digits.

Discovery: Discovered when completing unit testing. I built a variety of good and bad unit tests programmatically and used junit to weed out unit tests that were failing. Because the specific case that failed was a case with a 5 digit port number and the other cases were still passing as expected (including shorter port number), I knew the 5 digit port number was the issue. This allowed me to localize the issue to how the URLValidator was interpreting port numbers. Reviewing the regular expression pattern, I found it was only allowing repeating digits of length 1 - 3. .

For example:
http://www.google.com:80
Is valid:

While
http://www.google.com:50000
Comes back as invalid (both should be valid urls)

Cause:
In line 158 of “UrlValidator.java”, the regular expression pattern does not allow for a match on 5 digits. The repeating digit format that would be used to indicate a port number would begin with “:” and contain 1 to 5 digits. The regular expression assigned to PORT_REGEX only allows for 1 to 3 digits.

Original line of code:
```
    private static final String PORT_REGEX = "^:(\\d{1,3})$";
```

Rewritten line of code:
```
 private static final String PORT_REGEX = "^:(\\d{1,5})$";
```

Agan’s Rules:
I believe number 2 was especially helpful. This is not a type of URL, most people are accustomed to using. Most navigation online is done on the server side and most people are not actually typing port numbers into

their URLs on a daily basis.  Trying all types of cases is important as the more unusual ones are more likely to go without being noticed for longer and are harder to track down.