CS362 Final project - Part b

Jeremy Fischman

Huy Pham

Xisheng Wang

1. Description of methodology:

We were able to find four bugs using manual testing, partition testing, and programmatic testing. For manual testing, we built several sets of urls to test each part of a url incrementally. Our partition testing included a good case test, a bad case test, and several special test cases and boundary tests.

For debugging, we used several of Agan's principles. First, we used the divide and conquer principle, by testing possible causes of failure individually. Secondly, we also used the principle "make it fail" by generating tests that definitely produced buggy behavior. Third, we needed to "understand the system" at least at some level to help to localize the bugs. Finally the bugs were identified using Eclipse Debugger and code review. By placing breakpoints and keeping track of the value of key variables we were "keeping an audit trail", although our audit trail for this project was relatively rudimentary. Also, since we were debugging in part by doing a code review, this required us to occasionally "quit thinking and look," -- several times we needed to stop looking at the code for what we expected to see and notice instead what was actually there.

2. How we worked as a team, how we divided the work:

To divide the work, our team gave one partner the job of manual testing, one partner did partition testing, and one partner did programmatic testing. Then, our group debugged each bug individually and compared results to ensure we agreed on the cause of each bug. Once we agreed on the test results, we retrofitted and refactored each other's changes into our own code so that we all would work on the most current version of the program.

3. How we collaborated:

Our team scheduled our first meeting via Google Hangout to discuss about the project as well as how we plan to work and assign work for each member. From there, we collaborated mostly through emails. We shared documents through Google Drive and Google Docs. When a member of the team discovered a bug, we wrote up a bug report and notified the rest of the team members.

4. Some of our manual test cases

   URLs tested:
   https://www.yahoo.com/ return true
   http://yahoo.com return true
   3ht://www.yahoo.com return false
   http://255.34.235.2/ return true
   http://259.34.235.2/ return true
   http://www.google.com:34567 return false
   http://www.google.com:123/test1/ return true
   http://www.google.com:123/test1/?action=view return false

5. How we picked our partitions

For partition testing, our group decided that we could group all of the possible partitions as all of the possible schemes. By segmenting the testing this way, we have the ability to expand our testing as we see fit as well as creating different testing for each different scheme. In our testing, we decided to test the two main partitions, hyper-text transfer protocol ("http" and "https") and the file transfer protocol ("ftp"). These two transfer protocols have can have a significantly different syntax, since the ftp protocol could have a username and password in the URLs.

From here, our partition testing and our programmatic testing looked very similar. A series of URLs for each partition were generated by randomly picking URL components from an array. Each component had a boolean validity value, either true or false. A URL comprised entirely valid components produced a valid URL, and a URL that contained at least 1 invalid component was an invalid URL.

We then passed these assembled URLs to the URLValidator.isValid() function and checked that the return value matched the expected validity of the URL. If it did, the test passed, and if not this indicated a failed test.

6. How we built our program-based testing

Our program-based testing was very similar to our partition tests. Program-based testing was conducted by creating arrays of possible schemes, hostnames, ports, paths, and queries. Additionally, a second array of booleans was created for each URL component. A complete URL was created by choosing one of each URL component at random from the various arrays. Each array contained several valid and invalid components, and the corresponding booleans were selected from the boolean arrays. When a URL was created of entirely valid components its' validity was calculated by performing a logical "and" operation of each of the component booleans. If any of the components was invalid, then the entire URL was invalid. Finally, the URL and the validity value was passed back to the calling function.

This validity was compared to the validity returned by the URL Validator for the assembled URL. When they matched, the test was passed, and when they differed, this indicated a failed test.

7. All of our 4 bug reports. Include file name, line number, and debugging details.

<table>
<tr><td colspan="2">Title: Validation for IP address is incorrect.</td></tr>
<tr><td colspan="2">Reported by: Jeremy Fischman, Huy Pham, Xisheng Wang</td></tr>
<tr><td>Date Created: 3/11/16</td><td>Date Updated: 3/11/16</td></tr>
<tr><td colspan="2">Bug details:</td></tr>
<tr><td>Type: Bug</td><td>Status: Open</td></tr>
<tr><td>Priority: Major</td><td>Resolution: Unresolved</td></tr>
<tr><td>Affects Versions/s: None</td><td>Fix Version/s: None</td></tr>
<tr><td colspan="2">Components: None</td></tr>
<tr><td colspan="2">Labels: None</td></tr>
<tr><td colspan="2">Description: Valid IP address should be in a range from 0.0.0.0 to 255.255.255.255. However, the validator isValid() method return http://259.34.235.2/ as a valid Url.</td></tr>
<tr><td colspan="2">Debugging details: The failure was identified by manual test. The fault was manually tracking back to the method responsible for checking valid numerical IP address:
File name: inetAddressValidator.java
Function: isValidInet4Address()
Line: 96.
Problem description: Function should return false when the variable iIpSegment is larger than 255, but it currently returns true.</td></tr>
</table>

Title: UrlValidator does not accept number 34532 as valid port number.	
Reported by: Jeremy Fischman, Huy Pham, Xisheng Wang	
Date Created: 3/11/16	Date Updated: 3/11/16
Bug details:	
Type: Bug	Status: Open
Priority: Major	Resolution: Unresolved
Affects Versions/s: None	Fix Version/s: None
Components: None	
Labels: None	
Description: A port number is a 16-bit unsigned integer, thus ranging from 0 to 65535. However, the validator returns http://www.google.com:34567 as an invalid Url.	
Debugging details: The failure was identified by manual test. Eclipse debugger was used to localize its cause. **File Name:** URLValidator.java **Function:** Constant declaration of PORT_REGEX **Line:** 158: **Problem Description:** The error is in the regular expression that defines the variable. The current line, 'String PORT_REGEX = "^:(\\d{1,3})$";' matches any sequence of 3 digits. It should be: 'String PORT_REGEX = "^:(\\d{1,5})$";' This would match any sequence of 5 digits. This would cause another error, however, because it would allow port numbers greater than the maximum allowable (65535)... so an additional test must be added to check for this condition.	

Title: UrlValidator does not correctly validate query in a Url.	
Reported by: Jeremy Fischman, Huy Pham, Xisheng Wang	
Date Created: 3/11/16	Date Updated: 3/11/16
Bug details:	
Type: Bug	Status: Open
Priority: Major	Resolution: Unresolved

Affects Versions/s: None	Fix Version/s: None
Components: None	
Labels: None	
Description: The url http://www.google.com:123/test1/?action=view should be a valid url, but the validator returns false.	
Debugging details: The failure was identified by manual test. Since scheme, path and authority of this url have been validated from previous manual tests, we focused on the query part. Using Eclipse debugger, we identified the error line of code is in the isValid() method lines 446. **File Name:** URLValidator.java **Function:** IsValidQuery() **Line:** 446 **Problem Description:** The function returns the opposite truth value from what it should be returning when the query is not null. It should be: return QUERY_PATTERN.matcher(query).matches(); But currently the line reads: return !QUERY_PATTERN.matcher(query).matches();	

Title: UrlValidator does not correctly identify the valid host for ftp scheme when use with user and password.	
Reported by: Jeremy Fischman, Huy Pham, Xisheng Wang	
Date Created: 3/11/16	Date Updated: 3/11/16
Bug details:	
Type: Bug	Status: Open
Priority: Major	Resolution: Unresolved
Affects Versions/s: None	Fix Version/s: None
Components: None	
Labels: None	
Description: The url <ftp://jon:snow@foo.com:80/home> is a valid url but the validator returns false for this test.	
Debugging details: The failure was identified by ftp partition test. In this test, we validate different variations of ftp scheme by generating both valid and invalid urls for testing. Using	

Eclipse debugger, we identified the error line of code is in the isValidAuthority() method lines 377.
File Name: URLValidator.java
Function: IsValidAuthority()
Line: 377
Problem Description: When the ftp url has username and password included in the string, the parse function does not identify the correct hostname versus the username. As a result, the username is assigned as host name and this fails when validate in the domainValidator.