CS 362 SOFTWARE ENGINEERING II

Project: Part B

Fall 2015

GROUP MEMBERS:
Sharon Kuo, Tara Massey, Connor Peavey

METHODOLOGY

To develop a valid testing scheme we focused on the domain of inputs for the URL Validator. We attempted to break up each section of the domain by URL part, if it is a valid input, and then by the scheme parameter used when creating the new URLValidator object.

Once our domain was determined and our partitioning plan was developed, we focused on the various testing methods we could utilize to locate bugs and, if a bug was found, follow-up with further tests to indicate the problematic section of code. The testing methods we focused on were manual testing for valid and invalid inputs for a new URLValidator for each scheme option, partition tests for each part of the URL, and then random testing of known valid URL parts to create pseudo-random string combinations.

In all, it was our plan that the manual tests would locate bugs in a full URL. The partition tests would then indicate which part of the URL validation process was failing, in order to help localize the erroneous section of code. Once this was complete, random testing would test for combinations that we, due to constraints, did not do, to make sure that each correct URL part works when integrated with its other components.

MANUAL TESTING

To begin our manual testing of URLValidator, we called the isValid() method of URLValidator with different possible inputs, both valid and invalid inputs. Each set of manual tests was run with after a new UrlValidator was created.

- The first batch of inputs was tested on a new UrlValidator that is ALLOW_ALL_SCHEMES
- The second batch of inputs was tested on a new UrlValidator that is NO_FRAGMENTS
- The third batch of inputs was tested on a new UrlValidator that is ALLOW_LOCAL_URLS
- The last batch of inputs was tested on a new UrlValidator(0), the default.

.

To begin creating our inputs, we selected a known valid authority and tested valid schemes:

http://www.google.com	Expect: True	Actual: True
h3t://www.google.com	Expect True	Actual: True
://www.google.com	Expect False	Actual: False
http:/www.google.com	Expect: False	Actual: False
“ “ www.google.com	Expect True	Actual: False

Next, we tested various URL authorities with the valid scheme “http://”.

http://www.google.com	Expected: True	Actual: True
http://www.google~.com	Expected: False	Actual: False
htttp://www.google	Expected: False	Actual: False
http://www.google.au	Expected: True	Actual: True

http://broke.hostname.com	Expected: False	Actual: False
http://hostname	Expected: False(default)	Actual: True
http://1.2.3	Expected: False	Actual: False
http://	Expected: False	Actual: False

From these, we took valid passing tests from before and tested various ports for validity:

http://www.google.com:80	Expected: True	Actual: True
http://www.google.com:300	Expected: True	Actual: True
http://www.google.com:3000	Expected: True	Actual: False
http://www.google.com:30000	Expected: True	Actual: False
http://www.google.com:	Expected: False	Actual: False
http://www.google.com:-80	Expected: False	Actual: False
http:///www.google.com:80a	Expected: False	Actual: False

Valid answers from the previous tests were then taken and tested with various test paths:

http://www. google.com/test1	Expected: True	Actual: True
http://www. google.com/	Expected: True	Actual: True
http://www.google.comtest1	Expected: False	Actual: False
http://www.google.com//test1	Expected: False	Actual: False
http://www.google.com/..	Expected: False	Actual: False
http://www.google.com/../	Expected: False	Actual: False

We then added path options for testing to the valid answers from the previous set of tests:

http://www. google.com/test1	Expected: True	Actual: True
http://www. google.com/	Expected: True	Actual: True
http://www.google.comtest1	Expected: False	Actual: False
http://www.google.com//test1	Expected: False	Actual: False
http://www.google.com/..	Expected: False	Actual: False
http://www.google.com/../	Expected: False	Actual: False

Path options were then appended for testing:

http:/www.google.com/test1/test1	Expected: True	Actual: True
http://www.google.com/test1/	Expected: True	Actual: True
http://www.google.com/test1test1	Expected: True	Actual: True
http://www.google.com/test1//test1	Expected: False	Actual: False
http://www.google.comtest1test1	Expected: False	Actual: False

Valid answers from the previous tests were taken and tested with queries appended:

http://www.google.com?action=view”	Expected: True	Actual: False
http://www.gogle.com?action=view&hi=hello	Expected: True	Actual: False
http://www.google.com?action=	Expected: False	Actual: False
http://www.google.comaction=view	Expected: False	Actual: False
http://www.google.com??action=view	Expected: False	Actual: False

MANUAL TESTING ERRORS FOUND

www.google.com was determined to be invalid. Expected value was Valid, as a blank scheme should trigger a default scheme being used.

http://www.google.com?action=view was determined to be invalid. Expected value was valid, as it is set up in the required query state.

http://www.google.com?ation=view&hi=hello was determined to be invalid. Expected value was valid, as it was set up in the required query state.

http://localhost was determined to be valid for a default UrlValidator. This was expected to be true for ALLOW_LOCAL_URLS, but should not have been valid in the default state.

http://www.google.com:3000 and http://www.google.com:30000 were determined to be invalid. Expected value was valid, as ports go from 1 to 5 digits.

http://www.amazon.com /file was determined to be valid when clearly whitespace should not be allowed in a URL.

PARTITION TESTING

It was decided that we would partition the input domain into URL parts, each of which would be composed of valid and invalid input. The partitioning for manual tests was divided as follows:

Scheme:

Valid Scheme: http://	Expected: Pass
Valid Scheme: h3tp://	Expected: Pass
Fragment: ://	Expected: Fail
Fragment: :/	Expected: Fail
None: “ “	Expected: Pass

Authority:

Valid: www.google.com	Expected: Pass
Invalid: broke.hostname.com	Expected: Fail
Fragment: www.google	Expected: Fail
Illegal Characters: www.google~.com	Expected: Fail
None: “ “	Expected: Fail

Port:

Valid: :80	Expected: Pass
Valid: :3000	Expected: Pass
Invalid: :-80	Expected: Fail
Invalid: :80a	Expected: Fail
Fragment: :	Expected: Fail

Path:

Valid: /test1	Expected: Pass
Fragment: /	Expected: Pass
Fragment: test1	Expected: Fail
Additional: //test1	Expected: Fail
Non-Alpha: ..	Expected: Fail

Queries:

Valid: ?action=view	Expected: Pass
Fragment: ?action=	Expected Fail
Fragment: /	Expected: Fail
Additional: ??action=view	Expected: Fail

PARTITION TESTING ERRORS FOUND

Testing Partition Scheme, testing function isValidScheme:

http://	Expected: True	Actual: False

Testing Partition Authority, testing function isValidAuthority:

“ “ blank authority	Expected: False	Actual: True

Testing Partition Queries, testing function isValidQuery

?action=view	Expected: True	Actual: False

Testing Partition Port, testing function isValidAuthority:

:3000	Expected: True	Actual: False

PROGRAMMING BASED TESTING/UNIT TESTS

For the programming based testing we began by taking valid url parts from our partitions, creating arrays from each part. Then, in a loop, each url component is selected from the array in the proper order to form a valid address. The address is then tested until all combinations with the proper order are completed. Each address is tested using isValid(), and if false is returned the faulty url is stored in an array. These arrays will be printed at the end to notify us of addresses that caused errors.

Separately we decided to take two of the units inside of UrlValidator that were giving us the most trouble and unit test both of them outside of the whole system. In doing this we were

hoping to isolate bugs by determining if they resided in the parsing functions, or in the validation units themselves. Doing this is an important principle of Agan, specifically the divide and conquer. These tests provided invaluable insight that we would not have gotten if we had just tested them inside the whole system. It definitely made stepping through the test easier as well.

NAME OF YOUR TESTS

- **public void testManualTest()** - Manual testing of full addresses, both valid and invalid
- **public void testYourFirstPartition**() - Tested isValidScheme(), scheme partition
- **public void testYourSecondPartition()** - Tested isValidAuthority(), authority partition
- **public void testYourThirdPartition()** - Tested isValidAuthority() with added port, port partition
- **public void testYourFourthPartition()** - Tested isValidPath, path partition
- **public void testYourFifthPartition()** - Tested isValidQuery, query partition
- **public void testIsValid()** - Programming Based Testing, loop of known valid url parts in a valid combination, tested until all combinations are achieved. Failed addresses stored and printed.
- **public void testisValidQueryUnitTest()** - Similar to testYourFirstPartition, but eliminates the partitioning aspect, therefore creating a more focused test on the unit isValidQuery.
- **public void testisValidSchemeUnitTest()** - Like testisValidQueryUnitTest, this test eliminates the introduction of possible partition errors, and directly tests isValidScheme.

HOW DID YOU WORK TOGETHER AS A TEAM

Our team worked together efficiently and met all assigned tasks on schedule. We agreed to meet once per week via Google Hangouts video chat to discuss the schedule for the week, when each section was required to be completed, and talk about any problems we may have encountered.

For group collaboration, a shared Github was created so all members could work collaboratively on the same file. Additionally, all progress was recorded in a Google Document each week, so every group member could see what was completed, what needed to be completed, comment when there was a concern or question, and help each other. For issues that were more pressing, a Google Chat was created so we could communicate in real time.

Work was not explicitly divided up and assigned. Instead each week we assigned a certain number of units to be completed. Each group member worked on what they could as their schedule allowed, and pushed the results to the shared Github. Then, each group member posted what they were working on, their plan, issues that they encountered, and next steps to

the shared Google Document so the next team member could pick-up where they left off with relative ease.

USE OF AGAN'S PRINCIPLES

- Understand the System: Project Part A made the group familiar with the system at a base level. We then each spent time reading the notes from the programmer, walking through the URLValidator with the debugger and watches called on local variables. We worked to understand when each function was called, what it was intended to do, and how it impacted the overall result.

- Make it Fail: In our manual tests and our partition tests we purposefully put in code that was invalid. We wanted to make the code fail to ensure the failure was handled correctly, and adjusted our oracle to indicate when this was not the case. Similarly, we also included cases we knew would pass. We were at a partial advantage coming into this assignment, because we knew we would be given a buggy product. Because of this we knew some correct inputs would also cause a failure.

- Quit Thinking and Look: This principle was a real double edged sword. On one hand it was extremely helpful in bug finding, because we were able to simple step through each and every line of the code in this validator. On the other hand, when in search of a particular bug, we felt as though we were going down a rabbit hole from which we would never emerge. Because of this we needed to take a different approach on this method. Instead of simply stepping through the debugger a few times, and finding where an error was, it was necessary to do a lot of console printing, with more and more specific cases. Still looking, just looking differently.

- Divide and Conquer: Following our manual tests we tested valid and invalid partitions down the smallest component of the URL. Logically, if the address failed its test then this portion would help us narrow down the source of the problem by the partition. Additionally, when there was a failure we used the above as guidance for placing breakpoints in the debugger to narrow down the most likely area of code that is causing the problem. When we encountered errors in specific components of a URL we then created unit tests to ensure the error was not due to partitioning, but in the actual unit validators themselves.

- Change One Thing at a Time: Each time a new component was introduced for testing, we used valid components for all other url parts. This was to ensure that this principle was maintained, and we could isolate the problematic portion of the address.

- Keep an Audit Trail: All tested code was recorded with expected results, as well as the reason that particular team members was investigating with that route. This ensured that other team members could understand what was going on, why the code was being tested, and prevented duplication of test areas or a failure of documentation. All changes

to code were also tracked by hosting our project on github, which makes all changes available in an easy side by side format.

BUG REPORTS

Title: [URLValidator] Validator flags ports with 4 or 5 digits as invalid

Project: URLValidator
Version: 1.4

Type: Bug, major
Reported by: Sharon Kuo
Email: kuos@oregonstate.edu

Created: 11/23/15
Updated: N/A
Resolved: N/A

Type: Bug
Status: Open
Priority: Major
Resolution: Not Resolved
Affected Version: 1.4
File name: UrlValidator.java
Environment: Windows 7 x64, Eclipse Mars, Java 1.7

Description: The isValid() method incorrectly determines that URLs with port lengths of 4 or 5 digits are invalid. Our manual testing shows this bug when calling isValid() as follows:

```
System.out.println("http://www.google.com:80");
System.out.println(urlVal.isValid("http://www.google.com:80"));
System.out.println("http://www.google.com:300");
System.out.println(urlVal.isValid("http://www.google.com:300"));
System.out.println("http://www.google.com:3000");
System.out.println(urlVal.isValid("http://www.google.com:3000"));
System.out.println("http://www.google.com:30000");
System.out.println(urlVal.isValid("http://www.google.com:30000"));
```

For port 80 and 300, the expected values are true, and the actual values are true. However, for ports 3000 and 30000, the expected values are true, but the actual values are false.

Code Causing Bug:
URLValidator.java line 161

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

Title: NULL scheme not accepted by UrlValidator
Project: URLValidator
Version: 1.4

Reporter: Connor Peavey
Email: Connorpeavey@gmail.com

Created: 11/23/15
Updated: N/A
Resolved: N/A

Type: Bug
Status: Open
Priority: Major
Resolution: Not Resolved
Affected Version: 1.4
File name: UrlValidator.java
Environment: Windows 7 x64, Eclipse Mars, Java 1.7

Description: In isValidScheme line 336 states that if scheme is null, then it is a false scheme. However, null schemes are valid. This bug can be reproduce by submitting the string "www.google.com" to the isValid function in the UrlValidator class with any flags set.This bug was discovered while doing manual testing in the testManualTest function, and occured in all flag variations.

Code Causing Bug:
isValidScheme() line 336

Title: Acceptable queries fail validation in UrlValidator
Project: URLValidator
Version: 1.4

Reporter: Connor Peavey
Email: Connorpeavey@gmail.com

Created: 11/23/15
Updated: N/A
Resolved: N/A

Type: Bug
Status: Open
Priority: Major
Resolution: Not Resolved
Affected Version: 1.4

File name: UrlValidator.java
Environment: Windows 7 x64, Eclipse Mars, Java 1.7

Description: In isValidQuery line 314 the variable representing the query string is missing the '?' mark, therfore making all non-null queries invalid. This can be reproduced by submitting "http://www.google.com?action=view" as the argument for the function isValid. This works with any flags set in the creation of the UrlValidator object. This bug was discovered while doing manual testing in the testManualTest function.

Code Causing Bug:
isValidQuery() line 314

Title: Whitespace in passing URLs in UrlValidator
Project: URLValidator
Version: 1.4

Reporter: Connor Peavey
Email: Connorpeavey@gmail.com

Created: 11/23/15
Updated: N/A
Resolved: N/A

Type: Bug
Status: Open
Priority: Major
Resolution: Not Resolved
Affected Version: 1.4
File name: UrlValidator.java
Environment: Windows 7 x64, Eclipse Mars, Java 1.7

Description: In isValidAuthority line 377 removes all trailing whitespace following authority as a means of separating port. This allows non-valid URLs to pass, when they should instead fail. This can be reproduced by submitting "http:www.google.com /" as the argument to the function isValid. This works with any flags set in the creation of the UrlValidator object. This bug was discovered while doing manual testing in the testManualTest function.

Code Causing Bug:
isValidAuthority() line 337

DEBUGGING DETAILS

In order to localize the fault we began with the organization of our scheme. Our testing scheme was intended to divide and conquer, further localizing the area the fault would could be located in. At the narrowest point of localization, we could determine which portion of the URL was passing a validity check incorrectly.

Based on this information, we began our bug search in the indicated portion of code that handled the validity check for the partition. From there, we set up break points and watches with the Eclipse debugger to further isolate the line number.

Individual bug detail information is as follows:

Validator flags ports with 4 or 5 as invalid: After determining that there was a bug in the length of ports being recognized, we decided to look at the isValidAuthority() function. In this function, we examined the port matcher, and looked at the variables involved at this point in the function. We traced the issue back through PORT_PATTERN, utilizing watches and breakpoints, which resulted from PORT_REGEX. We saw that PORT_REGEX only allowed ports of length 1-3, which resulted in the bug we observed.

Whitespace in passing URLs in UrlValidator: Initial organization and secondary testing of the first partition indicated that the isValidSchceme() function was likely to contain the error. From there code was traced with the debugger, noting an incorrect validity on line 336.

Acceptable queries fail validation in UrlValidator: Initial test organization and secondary testing of the query partition indicated that the isValidQuery() function was likely to contain the error. From there code was traced and variables were examined. It was noted that the question mark was missing from declaration on line 314. Break points were then sent and the we tested the hypothesis while uses watches.

Whitespace in passing URLs in UrlValidator: Initial test organization and secondary testing of the authority partition indicated the isValidAuthority() function was likely to contain the error. Break points were sent and we stepped through, line by line, with the debugger. It was noted that at line 377, all trailing whitespace following the authority is removed as a means of separating the port. This is the likely cause of allowing non-valid URLs to pass validation.

FULL URLS TESTED

- google.com:9/?action=view
- http://google.com:100/test1?action=view
- google.com:9/
- http://google.com:100/test1

- http://google.com:80/test1?action=view
- www.google.com:9/test1?action=view
- http://www.google.com:9/test1
- www.google.com:9/test1?action=view
- google.com:80/?action=view
- www.google.com:80/
- http://ww.amazon.com
- h://www.amazon.com
- http://p.amazon.com
- amazon.com
- http://www.amazon.com?hamBlast=True
- http://www.amazon.com /file
- http://3.amazon.com
- www.google.com
- https://www.google.com
- http://w.google.com
- http://www.google~.com
- htttp://www.google
- http://www.google.au
- **http://**broke.hostname.com
- http://hostname
- http://1.2.3
- http://
- http://www.google.com:80
- http://www.google.com:300
- http://www.google.com:3000
- http://www.google.com:30000
- http://www.google.com:
- http://www.google.com:-80
- http:///www.google.com:80a
- http://www. google.com/test1
- http://www. google.com/
- http://www.google.comtest1
- http://www.google.com//test1
- http://www.google.com/..
- http://www.google.com/../
- http://www.google.comtest1
- http:/www.google.com/test1/test1
- http://www.google.com/test1/
- http://www.google.com/test1test1
- http://www.google.comtest1test1
- http://www.gogle.com?action=view&hi=hello
- http://www.google.com?action=
- http://www.google.comaction=view
- http://www.google.com??action=view