**URL Validator Final Project Part B**

**Manual Testing**

**Testing Methodology**

For our manual testing practices, we decided to implement a bit of white-box testing to try to learn what inputs were allowed, and what inputs would be faulty. We had discovered while looking over the code for the final project one or two bugs, but had decided to continue when creating our tests to create permutations to attempt to find anything we had missed. These permutations were covered in testing when we created tests with specific input partitions, as well as a set of random tests that generated port and ip numbers.

For our groups manual tests, we had decided to manually create both valid and invalid URLs that would test each if statement that resides in the isValid function in the URL validator.

For Example we take a valid URL, like

<http://yahoo.com>

And then we introduce some defect within the statement, such as a space in the middle

<http://yah> oo.com

This is to help pinpoint what statements are problems, and what ones are valid, it also helps single out the defective parts of the url that was provided.

**Valid URLs Tested:**

<http://www.amazon.com>

<http://www.go.com>

<http://www.yahoo.com>

**Invalid URLs Tested:**

<http://www.amazon.com//>

[http://go.33](about:blank)

<http://yah> oo.com

**Partition Testing**

For our partition tests, we had focused in on two locations within the code, the IP validation, and the port validation.

In the IP validation section, we focus on validating proper IP addresses, and disallowing segments of the IP address that are above, or below the threshold for the valid address. The threshold for each segment of an IP address ranges from 0 to 255, and we test negative values, and values such as 256 and 1101 to ensure that values above maximum are not considered valid. We also test edge cases, such as cases where every value is above and below the threshold.

For the IP testing, some examples of URLs that were used include:

<http://255.255.255.255>

[http://256.256.256.256](about:blank)

[http://-101.-111.-121.-131](about:blank)

In the Port validation section, we focused on validating ports that have proper values that lie within a threshold of 0 to 99999 as ports should allow a value with 1 to 5 digits. If a value falls into a negative region or has more than 5 digits, it should not be accepted.

<https://www.google.com:0/q/w/e/r/t/y/u>

<http://8.6.7.5:309/?h=j>

[http://www.google.com:-1/q/w/e/r/t/y/u](about:blank)

[http://8.6.7.5:-309/?h=j](about:blank)

**Test Names:**

Manual Test:

* File: UrlValidatorTest.java
* Method: testManualTest()

First Partition:

* File: UrlValidatorTest.java
* Method: testYourFirstPartition()

Second Partition:

* File: UrlValidatorTest.java
* Method: testYourSecondPartition()

Random Tests:

* File: UrlValidatorTest.java
* Method: testIsValid()
* Helper method: ipTest()
* Helper method: portTest()

**Bug Reports:**

Bug 1

* **Failure:**

Country Code for the US is not properly recognised

* **How Was it Found:**

Method ManualTestURL - testing main country codes such as .uk, and .us

* **What is the cause of the bug?**

In the country code array that begins on line 248 in the file domainvalidator.java, it does not contain the US country code

Bug 2

* **Failure:**

URL containing a query returns as false

* **How Was it Found:**

Method testYourSecondPartition - found while testing the second partition.

* **What is the cause of the bug?**

Line 446 of DomainValidator.java contains a negation before the LOCAL\_TLD\_LIST causing it to incorrectly display as invalid

Bug 3

* **Failure:**

URLs with more than 3 integers are not valid, when URLs should be allowed to have up to 5 integers.

* **How Was it Found:**

Method ManualTestURL - found while manually testing ports

Method testIsValid() - assertion failure

* **What is the cause of the bug?**

line 393 of UrlValidator makes a call to check if the port matches the given limitations, row 158 receives this call, and is set to only accept 3 digit port numbers PORT\_REGEX = "^:(\\d{1,3})$";

Bug 4

* **Failure:**

IP addresses with values containing a number greater than 255 are accepted

* **How Was it Found:**

Method ManualTestURL - found while manually testing ip inputs

Method testYourFirstPartion - assertion failure

Method testIsValid() - assertion failure

* **What is the cause of the bug?**

Inet line 94 of the file InetAddressValidator.java returns true if iIpSegment > 255

**Debugging**

**Provide at what line/lines in what file the failure manifested itself.**

**Bug 1**

Line 248 of DomainValidator.java: the country code array does not contain the US country code

**Bug 2**

Line 446 of DomainValidator.java: contains a negation before the LOCAL\_TLD\_LIST causing it to incorrectly display as invalid

**Bug 3**

Line 393 of UrlValidator: makes a call to check if the port matches the given limitations, row 158 receives this call, and is set to only accept 3 digit port numbers PORT\_REGEX = "^:(\\d{1,3})$";

**Bug 4**

Inet line 94 of the file InetAddressValidator.java: returns true if iIpSegment > 255

**Did you use any of Agan’s principle in debugging URLValidator?**

We used Agan’s 1st principle by looking over the source code and attempting to understand the purpose of individual methods and the overall execution of the program.

We had implemented rule number 2 of Agan’s principles, we had looked into our manually tested errors, and found that errors like the ip address allowing a 256 value was a consistent way to make the program fail.

We used Agan’s 3rd rule by using the debugger to find the location of the fault in the code before creating additional test cases.

We used Agan’s 4th rule by continuing to use the debugger to check each step of execution of a method containing a fault.

We used Agan’s 5th rule by only checking only one factor of a faulty test case at a time, such as by changing one field of a faulty IP address.

We used Agan’s 6th rule by recording defects and faults in the team google document as we found them.

We used Agan’s 7th rule by creating multiple tests that tested one method in multiple different ways to ensure program behavior was consistent.

We used Agan’s 8th rule by communicating with our team member when we encountered a fault or attempted to create test cases to assist in narrowing down a problem.

We did not end up using Agan’s 9th rule, as the instructions for the assignment had stated that developers would be the ones to fix the issue, after the testers had narrowed down what the issue was.

**Provide your debugging details for each bug**

**Bug 1**

Bug 1 was found in method ManualTestUrl by testing some select country codes such as .us and .uk. We looked at the source code for DomainValidator.java and found that there was no .us in the country code array.

**Bug 2**

Bug 2 was found in the method testYourSecondPartition, by testing an appended query to the url. This was found by manually testing, and finding that an assert had unexpectedly failed, after following the debug log, it had shown that the result of the error was a flag to accept a URL which was incorrectly set to false.

**Bug 3**

Bug 3 was found in the method ManualTestUrl by manually choosing some ports that we knew to be acceptable values, such as 65535. After finding these tests to fail when we hypothesized they should not, we traced the program execution to find that line 158 of UrlValidator.java only accepts a maximum of 3 values. This prevents valid ports, such as 11111, or 65535, from being accepted as valid parts of URLs.

**Bug 4**

Bug 4 was found in multiple different methods including ManualTestURL, testYourFirstPartion, and testIsValid. This was found by inserting values that were greater than the maximum allowed value for an IP address. This was found while creating and running manual tests, after receiving an unexpected result from testing 256 as a value in an IP field, I followed the debug log to find the error where a method to validate the maximum of an IP address was incorrectly set to allow numbers greater than 255

**Team Work**

**How Did We Work In The Team:**

We both did our own initial manual testing of the program, and then collaborated to write the final document and tests.

**Work Division:**

Nick designed the manual and partition testing methods including testManualTest(), testYourFirstPartition(), and testYourSecondPartition()

Sam designed the random testing methods including testIsValid(), ipTest(), portTest()

Both of us worked on the document.

**How We Collaborated:**

We used google docs and skype in order to collaborate on this project.