Final Project Part B:

Ellard Gerritsen van der Hoop- gerritse@oregonstate.edu
Shawn Seibert- seiberts@oregonstate.edu
Suyana Lozada- slozada@oregonstate.edu

Spring 2016

Introduction:

For the second part of the project, we were tasked with writing manual tests, partitioning tests, and unit/random tests for the URL Validator program provided to us. Once these tests were created, we determined if there was a bug in the URL Validator code. If a test failed, we debugged the provided code to try and determine where the error did in fact occur. If a bug was found in the code, we developed a bug report with various details on how and why the bug is affecting the code.

Creating and Implementing Tests:

Our method for manual testing involved prompting the user with a url, and then stating whether the URL will return True or False. We used urlVal.isValid() function to compare against a boolean variable. If the URL is valid, and the boolean value is true, then the test should pass. If the URL is invalid, and the boolean value is set to false, then the test should still pass. However, if the URL is invalid and the boolean value is set to true, the test should fail.

URLS for manual testing and results include:

http://www.amazon.com	Valid URL | Test Passed
https://www.google.com	Valid URL | Test Passed
https:////www.google.com	Invalid URL | Test Passed
ftp://www.google.com	Valid URL | Test Passed
fttp://www.google.com	Invalid URL | Test Failed as correct URL
wxxj://www.google.com	Invalid URL | Test Failed as correct URL

After completing manual testing, we moved onto partitioning testing. The partitioning tests involved us breaking down each portion of the URL into their own separate entities. For example, one partition is an array of TLD's (com, edu, gov, etc) that are both valid and invalid. Another parition would only deal with Port Numbers (:80, : 1000, :33s3), both valid and invalid. Within each parition tests, we tested each poriton of the URL with the isValid() function. If the partition was valid, then the test should return as a valid URL. If the partition was invalid, then the test should return that is an invalid URL. However, if a valid URL was used, and an invalid URL was returned, then we knew there was a bug. An example of the array used for partition testing is shown below.

```
static ResultPair[] ports =
        {
                new ResultPair(":1000", true),
                new ResultPair(":999", true),
                new ResultPair(":0", true),
                new ResultPair(": ", false),
                new ResultPair(":Zero", false),
                new ResultPair(":755", true),
                new ResultPair(":65534", true),
                new ResultPair(":33s3", false)
        };
```

Lastly, we conducted random testing, with 50 iterations, of an entire url string. We created various arrays that included each portion of a URL. From there, we concatenated the URL randomly which would then be tested with the test isValid() function. If the randomly combined URL was valid, then we expected the isValid() funciton to return a valid URL result. If the created URL was invalid, then we expected the isValid() function to return an inValid URL result. Below is a sample of our code for testing a valid URL:

```
if(rand.nextInt()%2 == 0){
                    expected = true;
                String scheme = goodSchemes[part1];
                String authority = goodAuthority[part2];
                String port = goodPort[part3];
                String path = goodPath[part4];
                String query = goodQuery[part5];
                result = urlVal.isValid(scheme + authority + port + path + query);
                System.out.println(scheme + authority + port + path + query);
                System.out.println("Return:" + result +" Expected:"+ expected);
                if (expected != result){
                        System.out.println("\n\tERROR: Expected result does not
                         match\n");
                }
                System.out.println("");
```

Using Agans Principle:

With the first part of this project requiring us to figure out how the URL validator works by stepping through the entire validation process, we implemented Agan's first rule of "Understanding the system". By knowing how the system works, we were able to better determine how and what tests need to be created to determine if the URL Validator works correctly. This methodology would fall under Agan's second rule of "Make It Fail".

Move on to the actual testing, we implemented three different methods to test the code. We started with manual tests to determine if valid and invalid URL's would pass as expected. From there, we moved onto partitioning tests. These partition tests, which uses Agan's Rule #4 "Divide and Congquer", allowed us to pin point exactly where the bug was located. Lastly, we utilized random tests to randomly create a URL that would either be a valid or invalid URL.

When a bug was found during the testing, we set up break points at locations that we thought were the cause of the bug. At that point we began documenting when and where the bug would occur. This process of documentation fell under Agan's Rule #6 "Keep an Audit

Trail". We used this documentation to recreate the same bug in case we missed details when writing the bug reports.

Group Collaboration:

For this project, we utilized Google Hangouts for any group meetings and discussions. By utilizing this method, we were able quickly come up with a plan to break up this project and set up a timeline for completion. We broke up the testing amongst teammates so one person worked on manual testing, another worked on partitioning, and the remaining teammate worked on unit testing. Once testing was completed, we went through each failed test to determine what was causing the bug. From there, each teammate selected a bug(s) to right a report on. Lastly, each team member reviewed the other team members reports ensuring they were correctly written.

Bug Reports:

Bug Report 1:

Title: Incorrect returned value when testing URL Query
Product:
Eclipse Java EE IDE
Version: Mars.2 Release (4.5.2)
Build id: 20160218-0600
Classification: Serious bug that prevents the URL Validator from working correctly
Platform: Windows 10
Can it be reproduced: Every time
Description: When creating a query to be tested, a correct query comes back as false.
What's the error: There was no error displayed when executing the test.
Supporting information: When running our partitioning tests against various valid and invalid queries, the valid queries would return as false. After debugging the code, the issue appears to be coming from Line 447:
return !QUERY_PATTERN.matcher(query).matches()
Filename of Bug: UrlValidator.java
Steps to reproduce:
Step 1: Create a valid URL with query:
http://www.oregonstate.edu/login?userName=cs362
Step 2: Run the URL Validator program against the URL in step 1

Step 3: The URL Validator should return that this URL is invalid

Expected Results: If the query is valid, and the remaining URL is valid, then I expect

Actual Results: Any query, whether valid/invalid, return false

Debugging: When placing a breakpoint at line 314 of UrlValidator.java

```
if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
        return false;
 }
```

We can step into the function which is located at line 441. By continuing through each line

of the function, we can see that this function only returns true when the query is empty.

When it gets to line 447, a boolean value returns false for any string passed to it.

Work Around: If there are no queries added to the URL, it will pass the URL Validator test

Attachments: N/A

Contact: Name: Shawn Seibert Email: seiberts@oregonstate.edu

Bug Report 2:

Title: Correct Port number produces negative result

Product:

Eclipse Java EE IDE

Version: Mars.2 Release (4.5.2)

Build id: 20160218-0600

Classification: Serious bug that prevents the URL Validator from working correctly

Platform: Windows 10

Can it be reproduced: Every time

Description: When creating a with a URL greater than 999 causes the validator to produce an invalid URL result

What's the error: There was no error displayed when executing the test.

Supporting information: When running our partitioning tests against various valid and invalid ports, the valid ports would return as false. After debugging the code, the issue appears to be coming from Line 158 of the UrlValidator.java file.

Filename of Bug: UrlValidator.java

Steps to reproduce:

Step 1: Create a valid URL with a port number greater than 999:

http://www.oregonstate.edu:1000

Step 2: Run the URL Validator program against the URL in step 1

Step 3: The URL Validator should return that this URL is invalid

Expected Results: If the port is valid, and the overall URL is valid, then I expect the URL Validator to return true

Actual Results: A port number greater than 999, whether valid/invalid, return false

Debugging: When placing a breakpoint at line 223 result = urlVal.isValid(combinedURL); of UrlValidator.java, we can step into this function. By continuing to step through the code, we end

up at line 393: if (!PORT_PATTERN.matcher(port).matches()). By breaking down this code, you will see PORT_PATTERN. When examining this declaration at line 159, we see that PORT_PATTERN actually equals Pattern.compile(PORT_REGEX). By further examining PORT_REGEX on line 158 equals "^:(\\d{1,3})$", we can see that only decimal values from 0 to 999 are allowed based on 1,3.
Work Around: There are no workarounds for the port numbers
Attachments: N/A
Contact: Name: Suyana Lozada Email: slozada@oregonstate.edu

Bug Report 3:

Title: Incorrect Scheme used returns a valid URL
Product:
Eclipse Java EE IDE
Version: Mars.2 Release (4.5.2)
Build id: 20160218-0600
Classification: Serious bug that prevents the URL Validator from working correctly
Platform: Windows 10
Can it be reproduced: Every time
Description: When creating a with a URL various letters
What's the error: There was no error displayed when executing the test.
Supporting information: When running our partitioning tests against various valid and invalid ports, the valid ports would return as false. After debugging the code, the issue appears to be coming from Line 340 of the UrlValidator.java file.
Filename of Bug: UrlValidator.java
Steps to reproduce:
Step 1: Create a valid URL with a scheme fttp://:
fttp://www.oregonstate.edu
Step 2: Run the URL Validator program against the URL in step 1
Step 3: The URL Validator should return that this URL is valid
Expected Results: If an invalid scheme is used, the URL validator should return an invalid result
Actual Results: Using an invalid scheme returns a valid URL.
Debugging: When placing a breakpoint at line 165 result = urlVal.isValid(combinedURL); of UrlValidatorTest.java, we can step into this function. By continuing to step through the code, we end up at line 340 which appears to be the location of where this bug is:

```
if (!SCHEME_PATTERN.matcher(scheme).matches()) {
        return false;
}
```

When hovering over SCHEME_PATTERN, we see the following values below:
^\p{Alpha}[\p{Alnum}\+\-\.]*
It is within the SCHEME_PATTERN that the bug is causing the issue with invalid schemes producing valid URLS

Work Around: There are no workarounds for the port numbers
Attachments: N/A
Contact: Name: Ellard Gerritsen van der Hoop Email: gerritse@oregonstate.edu