Will Thomas
Isaiah Perrotte-Fentress
Timothy Robinson
3/14/2016

URLValidator Testing Summary

Methodology

For Manual Testing: Testing manually involved considering a subset of both valid and invalid URLs. Since it would be both time consuming and inefficient to hard code code an exhaustive list with no other organization we tried to come up with a list of inputs that were indicative of obvious, high level, and unique problems. For valid URLs we used inputs such as:

"http://www.amazon.com",
"https://www.google.com",
"http://ww.google.com",
"ftp://127.0.0.1",
"https://oregonstate.instructure.com/courses/1568425",
"http://eecs.oregonstate.edu/current%20students"

For invalid URLs we used inputs such as:

"http://www.google.com:123r",
"https:///www.google.com",
"http//www.oregonstate.edu",
"http:\\www.amazon.com"

To test we loaded both sets into arrays and then iterated through them running the isValid() method on each index.

For Input Partitioning: Input partitioning was done across the structure of a URL itself. Since URLs have very clearly defined structure including <scheme>://<authority> <port> <path> <query>, it made sense to test each element one at a time, while controlling the rest of the input with a correct value. This led to a series of individual tests for each element. For example, testYourFourthPartition() is focused on testing queries. To do so we created a controlled “correct” URL, "http://www.google.com", and then proceeded to add various test values onto the end of it. In this case we tested values such as "?action=edit&mode=up" and then ran through isValid(). For each partition, we tested both valid and invalid forms of that partition. For example, in testing the paths partition of the URL, we used “/test1” and “/test1/pics” as valid paths and “/..” as an invalid path.

Partitioning this way makes sense, since UrlValidator tests each element of the URL individually with its own method. This way you can isolate any logical errors at each step.

For programmatic testing the methodology was to ensure we had a robust set of inputs, and then let the system iterate over all of them. During the iteration we would compare the result of our Oracle (see if all parts of the URL are valid), and compare that against the result from isValid(). Since the URLs are generated using nested for loops that hit all possibilities, similar URLs are all grouped together. This makes debugging easier, because you will see a trend of failures that allows you to root cause which part of the URL is causing the failure.

Finally, at the end I wanted a summary to see if progress was being made for the developers when they are fixing their code. The summary tells you how many tests failed, and the percentage of all the tests run. As a developer running tests after implementing changes, I could quickly see the impact they had.

Team Responsibilities

Isaiah Fentress: Handled part three (the programmatic testing). For this portion designed one unit test that covered all of the different portions of the URL at once. Created ResultPairs that contained an adequate number of test examples using online resources. This provided a robust test bank that would locate almost any error (especially in TLD or protocol).

Will Thomas: Worked on manual testing and input partitioning. This involved the methods in question along with hard coding the values, as well as debugging two of the found bugs.

Tim Robinson: Refactored and edited test code. Commented and formatted test output. Contributed to and edited report.

Will + Isaiah: Worked on report together utilizing google docs to share bug information. We utilized google hangouts to meet and communicate.

Bug Reports

Bug 1:
Title: Port Validation Failure
Reporter: Will Thomas
Created: 3/13/2016
Type: Bug
Status: Open
Priority: Minor
Resolution: Unresolved
Labels: URLValidator

Description:

UrlValidator is failing when passed certain port values at the end of URLs. When I submitted a URL with port 80 it passed, but it failed port 8080. In the file **UrlValidator.java** on lines 158 and 159 port regex only allows a numeric value with 3 integers when ports can have more than that.

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

Bug Report Snippet:
Testing Valid URL:
TEST PASSED: "http://www.amazon.com:80"
TEST FAILED: "http://www.amazon.com:8080"

Suggested Fix
Adjust the port regex so that it recognizes all appropriate patterns.

What is the Failure:
The validator does not have all port sizes in consideration.

How did you find it:
During manual testing I noticed the above response in my valid URL list. I was unsure why the 2nd URL failed when the rest of the URL was valid. After running the debugger everything looked fine but I saw that the port validation was failing for 8080.

Bug 2:
Title: Query Validation Failure
Reporter: Will Thomas
Created: 3/13/2016
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Labels: URLValidator

Description:
UrlValidator is failing when passed any valid query value at the end of a URL. This failure seems independent of any specific value in the query. The specific error was identified in UrlValidator.java on lines 441 - 447. The function isValidQuery's logic is set up in such a way that it returns true when the query is null and the opposite of whatever the matcher returns otherwise.

```
protected boolean isValidQuery(String query) {
    if (query == null) {
        return true;
```

```
        }

        return !QUERY_PATTERN.matcher(query).matches();
    }
```

Bug Report Snippet:
Testing Query
TEST FAILED. Expected VALID for URL: "http://www.google.com?action=view"
TEST FAILED. Expected VALID for URL: "http://www.google.com?action=edit&mode=up"
Query Test Failed

Suggested Fix
Logic for isValidQuery needs to be modified in such a way that it imitates the isValidScheme function:

```
    protected boolean isValidScheme(String scheme) {
        if (scheme == null) {
            return false;
        }

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

        return true;
    }
```

What is the Failure:
Any valid query is evaluated as false because of this line:
`return !QUERY_PATTERN.matcher(query).matches();`

How did you find it:
During Input partitioning I noticed that all of the query values I input were returning false regardless of what they included. After running the debugger I stepped into the isValidQuery method and saw the incorrect logic in question.

Bug 3:
Title: Country TLD Failure
Reporter: Isaiah Fentress
Created: 3/13/2016

Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Labels: URLValidator

Description:
In the **DomainValidator.java** file on line 248 there is a String array called **COUNTRY_CODE_TLDS.** This contains all of the valid TLDs for countries. On line 357 in the array it ends abruptly with Italy ("it"). This means that all countries that start alphabetically after "it" will be reported as failed by the isValid() test. Additionally, the code has a trailer comma which is incorrect syntax. This might be causing additional issues with this array.

```
   private static final String[] COUNTRY_CODE_TLDS = new String[] {
      "ac",                 // Ascension Island

      "ad",                 // Andorra
      "ae",                 // United Arab Emirates
      "af",                 // Afghanistan
      "ag",                 // Antigua and Barbuda
         … // to shorten snippet
      "in",                 // India
      "io",                 // British Indian Ocean Territory
      "iq",                 // Iraq
      "ir",                 // Iran
      "is",                 // Iceland
      "it",                 // Italy

   };
```

Bug Report Snippet)examples that caught this error):
TEST FAILED 695161: URL: https://www.drought.sn/somepath1?request=answer Expected: true Result: false
TEST FAILED 695162: URL: https://www.drought.sn/somepath1?one=ten&two=eleven Expected: true Result: false
TEST FAILED 695163: URL: https://www.drought.sn/somepath1 Expected: true Result: false
TEST FAILED 695164: URL: https://www.drought.sn/somepath1? Expected: true Result: false

Suggested Fix: Add all country codes per this wikipedia article:
https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains
Ensure proper syntax by ending the last element without the comma.

What is the Failure:
The system does not have an exhaustive list of TLDs that are valid. As such, isValid() was reporting false when it should have been reporting true.

How did you find it:
I was able to find this bug by using a complete list of all valid top level domains in my test script. I was noticing the second half of my tests were taking longer to run than the first half (meaning more failures). I noticed several tests fail that were correct items. I investigated the code that controls what TLDs are allowed, and noticed that it was missing numerous items off this list: https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains

Agan's Principles Used

Quit thinking and look
I was searching for a root cause for a while on why a certain subset of items were failed (the ones near the end with higher alphabetical TLDs). I eventually just decided to look at the isValid() code and immediately noticed that country TLDs after Italy are missing. And in fact, the array has an error because it ends with a comma without another element, which might be causing other issues as well.

Divide and Conquer
This principle was integral for this assignment. Though divide and conquer was an important principle for each part of this assignment, the partitioning section of our testing straightforwardly relied on this principle. Dividing our input space into separate pieces and exhaustively testing those smaller pieces was fundamental to our approach.

Change One Thing at a Time
This principle was also very important to our assignment, as in all debugging experiences we've had. The potential for unpredictable results rooted in the interaction of two or more variables changed at a time would make it significantly more difficult to debug a program than sticking to one independent variable.