URLValidator.pdf: Final report for Final ProjectPart B

Team Members
Dane Schoonover -schoonod@oregonstate.edu
Aleksandr Balab - balaba@oregonstate.edu
Brett Irvin - irvind@oregonstate.edu

Methodology of testing:

We started by checking the variety of inputs as well the boundary cases. Afterwards we performed some manual testing in which we hardcoded known valid URLs and invalid ones and passed it to isValid(). Our next step was to partition the different elements of the input URL string to narrow down the location of bugs. Finally, we built many URLs by combining different parts into a complete URL to input into isValid().

The URLs that were used in manual testing:

Valid:

- http://www.hotmail.com
- http://www.facebook.com
- https://www.bing.com
- https://www.bing.com/maps/
- http://www.bing.com:80

Invalid:

- htt://www.facebook.com
- https://www.bing.com/ maps
- http://www.bing.com:111111111111
- http://www.amazon.123
- Empty string

Reason for choosing your partitions:

Our partitioning is based on the components of a URL. The first partition test (testPartition1()) tests randomly generated URLs and then is compared to standard URL form, we used ALLOW_ALL_SCHEMES flag to allow all schemes to be considered valid on top of default ones. We used “bing.com” as a strandard authority and domain suffix.

In our second partition test (testPartition2()) we tested the authority component of the randomly generated URL. We ran quasi-random tests based on known valid inputs and afterwards we tested the dotted decimal URLs. We generated authorities with schemes and domains that are known to be valid inputs.

Provide names of tests:

- testManual()
- testPartition1()
- testPartition2()
- testPorts()
- testIsValid()
    - testIPv4();
    - testUrl();
    - testASCII();
    - testIsNull();
    - testIsValidQuery();
    - testIsValidPath();
    - testIsValidFragment();

How work was divided in your team:

We agreed to divide workload equally among team members. Each team member provided code and tests, and each team member contributed to the discussion and submitted one bug report.

How teammates collaborated:

Mainly the team met via Google Hangouts to discuss about the progress each team member made. This was very beneficial to be able to verify which tasks remained to be completed. The majority of our discussions were related to trying to understand the code so that we could brainstorm different methods to create our tests. After dividing the work among the team, we continued to meet once a week. Feedback on a member's particular test was assured through push/pull through Github and running the tests in our own IDE. This, along with our code coverage tests (see below) allowed the group to determine if any improvements in any tests were required. We shared various resources that were related to the identification of a URL to help with the construction of our tests.

Methodology:

To assess the overall quality of our unit tests, we looked at how much code coverage we were getting. This was done by downloading and installing EclEmma, a popular Eclipse plugin. We referred back to this plugin while we were developing our tests, both to determine how good our current tests were and to identify areas that needed improvement. We are getting an estimated 90% of code coverage with our current test files, which we felt was sufficient for moving forward with the next steps in the project.

The majority of our tests used assertion statements or printed test results to the console. When results were being output to the console, we set the tests up to print the standard "Expected" versus "Actual" that we have used in other assignments throughout the class. When we needed to localize an error or restrict it to a single function, the tests were set up as JUnit tests in Eclipse; this allowed functions to be run individually, which was much faster in narrowing down the location of some bugs.

Bug Report #1:

Impact/priority: Significant

Location: UrlValidator.java, line 446

Description: The isValidQuery method returns incorrect results. The function should return true if the query is null or is valid. The code dealing with the null query is correct, but the return statement at the end of the function is incorrect. The current return statement will always return the opposite boolean of what is correct (if the return should actually be true, the current return statement will always return false, and vice versa), meaning it will flag valid URLs as invalid.

Found by: Some of the testIsValidQuery tests were failing (for example: 'http://www.bing.com?foo=bar&bar=foo' was being flagged as false instead of true). Reviewing the function, we had two very similar URLs that were being tested, 'http://www.bing.com?foo=bar&bar=foo' (which failed the test), and 'http://www.bing.com?=foo&bar=foo' (which passed). Noticing that there were very few differences between these two URLs led us to the conclusion that either we had a random error that was only triggering for certain inputs, or that we had a more serious error that was flagging most of our URLs incorrectly. Since the only difference between the two URLs above is in the query string, the error didn't seem to be the result of an incorrect regex statement, so the next step was to review the isValidQuery function for possible logic errors, which is where the error on line 446 was discovered.

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

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

Debugging Details (pasted from Eclipse console):

```
Testing queries...

First testing that http://www.google.com
Result: true

http://www.bing.com?action=delete
Expected: true
Actual: false

http://www.bing.com?foo=bar&bar=foo
Expected: true
Actual: false
```

Recommended fix: The return statement should be modified to `return QUERY_PATTERN.matcher(query).matches();` to produce the intended results.

Agan's Rules: The principle used in this case was Agan's rule #3: "Quit thinking and look." We had two similar URLs to compare, one which passed the test, and one which failed (though both were actually valid URLs). By simply looking at the differences between these URLs, it was fairly straightforward to track down the error. The only difference between the two URLs was in the query string, so that seemed like a good place to start. On the surface, this bug could have been located almost anywhere, but by taking a step back and actually looking, it became obvious where the problem was.

Bug Report #2:

Impact/priority: Moderate, since the allowLocal flag is required to trigger the condition

Location: DomainValidator.java, line 139

Description: Similar to the bug listed in Bug Report #1, the isValid function of DomainValidator is returning an incorrect boolean value. The isValid function takes the domain as a parameter and if the allowLocal flag is set, will evaluate an invalid domain as valid. This is because the hostname regex match statement is set to evaluate to 'true' even if the hostname is invalid. This bug is somewhat localized, since you have to have the allowLocal option enabled to reach the incorrect code segment.

Found by: This bug was found while reviewing the isValid function while trying to track down another error.

```
public boolean isValid(String domain) {
    String[] groups = domainRegex.match(domain);
    if (groups != null && groups.length > 0) {
        return isValidTld(groups[0]);
    } else if(allowLocal) {
        if (!hostnameRegex.isValid(domain)) {
            return true;
        }
    }
    return false;
}
```

Recommended fix: The `if (!hostnameRegex.isValid(domain)) {return true;}` statement should be modified to if `(hostnameRegex.isValid(domain)) {return true;}` to produce the intended results.

Agan's Rules: The rule that applied here was #1: "Understand the system." By going over the isValid function line by line and making sure to understand what each line of code was doing, a bug that would have otherwise been easy to overlook was found. It seems likely in retrospect that our tests would have either missed this bug entirely or would have thrown a single error that would have been easy to miss. In this case, understanding the system was critical because it forced us to review the code line by line until we were familiar with how the validator was coded, and how other bugs we had found had been introduced (in fact, this bug is similar to one found in Bug Report #1). After reviewing so many of the other functions in the validator and having an idea of where and how bugs could be introduced, it was much easier to quickly spot when something was out of place.

Bug Report #3:

Impact/priority: Substantial

Location: InetAddressValidator.java, line 94

Description: There is an error in the isValidInet4Address method. If any segment of an IPv4 IP address is greater than 255, the function will still return true, indicating that it is a valid URL.

Found by: This bug was found completely by accident. I had noticed what I thought was an inconsistency with port numbers, and was checking into that. While I was looking over the various files to find out where the ports were being evaluated, I happened to open InetAddressValidator. I noticed that towards the end of the isValidInet4Address function, the code for evaluating IPs would return true/valid if an IP segment was greater than 255.

```
try {
            iIpSegment = Integer.parseInt(ipSegment);
        } catch(NumberFormatException e) {
            return false;
        }

        if (iIpSegment > 255) {
                return true;
        }
}
```

Debugging Details (pasted from Eclipse console):

Expected = False: http://142.215.278.467
Expected = False: http://415.355.127.463
Expected = False: http://276.400.37.51
Expected = False: http://431.132.123.317
Expected = False: http://196.25.39.432
Expected = False: http://296.385.2.239
Expected = False: http://172.378.172.210

Recommended fix: The `if (iIpSegment > 255) {return true;}` statement should actually return a false boolean.

Agan's Rules: Several of Agan's rules could apply here, but the two that are most relevant to finding this bug are #1: "Understand the system," and #7: "Check the plug." This IP address bug was found while trying to track down port number tests that were failing. Trying to find the port number bug required going back over the validator files and finding each location where port numbers were actually being tested (understand the system), and, when no obvious errors were found, going through other related functions to see if they could possibly affect port numbers (check the plug--question your assumptions). Therefore, this IP address bug was found by combining two of Agan's rules.