Hantao Ling
Arthur Shing
CS 362 Spring 2017

Final Project: Part B

Testing Methodology

For our testing, we went through three stages: (1) Manual Testing, (2) Partition Testing, and (3) Unit Testing.

Manual Testing – For manual testing, we tried to test the URL "http://www.google.com/" with varying modifications. We added paths, removed different portions of the URL, and added extra portions to the URL. In this way, we were able to test both valid and invalid URLs in our manual testing. Some examples of the URLs we tested were:

- "www.google.com"
- "www.google.com//somefile"
- "http://www.google.com:-1"

Partition Testing – For partition testing, we partitioned the URL into five components: protocol, domain, port, path, and query. In each of these tests, we tested a valid and invalid version of the component and left the other four components as valid. We then recorded which of the tests failed. Some examples of the invalid URLs we tested were:

- Protocol: "http//www.oregonstate.edu"
- Domain: "http://jjiorewafli"
- Port: "http://www.oregonstate.edu:9999"
- Path: "http://www.oregonstate.edu/.."
- Query: "http://www.oregonstate.edu????lol"

We found that the valid port test failed as well as the invalid query test.

Bug Report

Bug 1 – Query Bug:

The query was wrong in all valid cases. We found the bug through our partition tests. When testing the query partition, it gave us an error on valid query cases, which prompted us to check the code regarding the query. From analyzing the code, we saw that the isValidQuery() function was checking to see that the query didn't match the valid query pattern instead of matching the query pattern.

Bug 2 – Port Bug:

The port partition test tested for port numbers ranging from 1 to 5 digits in length. From this test, we saw that the isValid() function was failing to correctly identify URLs with 4 digit ports. Based on this error, we checked how the function checked the port segment and found that the Port Regex only looked for port numbers up to 3 digits in length.

Bug 3 – Path Bug:

From our unit testing, we found that the isValid() function was unable to recognize an invalid path segment. In the unit test, we iterated through a large set of valid and invalid path segments and saw that the isValid() function consistently failed to identify a path with “/#” within it as an incorrect path segment.

Debugging

Bug 1 – Query Bug (line 446 in UrlValidator.java):

We were able to use the Eclipse debugger to find the Query Bug. The exact issue was that there was a not character (“!”) in front of the line “QUERY_PATTERN.matcher(query).matches()”, which defined what the function thought a query should look like. The “!” should not be present. Thus it evaluated every query of exactly opposite of what it should have been. The bug was found on line 446 in the UrlValidator.java file.

Bug 2 – Port Bug (line 158 in UrlValidator.java):

We used the Eclipse debugger to find the location of the Port Bug on line 158 in the UrlValidator.java file. The exact issue was that the variable PORT_REGEX, which helped define the PORT_PATTERN variable, was set to only validate port number up to 3 digits in length instead of 5 digits. The line in question was ‘PORT_REGEX = “^:(\\d{1,3})$”’, which should have been ‘PORT_REGEX = “^:(\\d{1,5})$”’ to account for 5 digit port numbers. The bug was found on line 158 in the UrlValidator.java file.

Bug 3 – Path Bug (line 310 in UrlValidator.java):

We used a combination of the Eclipse debugger and each of our three test functions (manual testing, partition testing, and unit testing) to find the exact line number that the path error was being generated on. We were able to use the debugger to find the cause of the bug (UrlValidator believes that “/#” is a valid path when it is not) by following it into the UrlValidator file. On line 310 in this file, we were able to watch the path variable being passed into the isValidPath() function and saw that when “/#” was part of the path, the “#” character was not being passed in as part of the path. Because of this, the function only thinks the path is “/” instead of the full “/#” path.

Team Work

Our team operated on the basis of splitting up the work into manageable chunks to be delegated to each member. Specifically, we divided the work into writing the report and writing the code between two members. However, in both cases, both of us were fully involved in contributing to the work. While one member physically typed the code up, the other member acted as a second pair of eyes and provided insight into the code being written. This was simple to do because both of us live in Corvallis, and we were able to meet up in person to work on the project. The other member worked on typing out the report and organizing the work. Again, this was a collaborative task because even though one person did the typing, the other always looked through the report to make edits and provide insight on writing style and organization. In this way, as a team, we were able to delegate work but still be fully involved in all aspects of the project.