CS362 - Final Project Report

Amy Dobbs, Andrew Gremmo, and Spencer Winlaw

Methodology of Testing

In this project we were asked to test the URLValidator program as a group. This is a program which we were unfamiliar with when we began. As such our task was to study the URL Validator code and develop an effective testing methodology. We approached the testing in three different ways - manual testing, partitioning, and automated unit testing. Each one is discussed distinctly below.

Manual Testing

We decided to begin our testing process with manual testing. We utilized this method primarily to familiarize ourselves with the URLValidator program. Each test consisted of creating a test case (a URL) and call to the isValid() function with that URL. We then analyzed the result to see whether it was what we were expecting. In our exploration we tested a variety of URLs. We provide a sample here:

```
System.out.println("http://www.amazon.com"));
System.out.println(urlVal.isValid("http://www.amazon.na"));
System.out.println(urlVal.isValid("http://www.amazon.za"));
System.out.println(urlVal.isValid("http://256.256.256.256"));
System.out.println(urlVal.isValid("http://localhost/"));
System.out.println(urlVal.isValid("ftp://localdomain/"));
System.out.println(urlVal.isValid("http://www.amazon¢.com"));
System.out.println(urlVal.isValid("com.amazon.http://"));
System.out.println(urlVal.isValid("amazon.http://com"));
System.out.println(urlVal.isValid("this_is_not_a_url"));
```

Some of the things we were interested in while performing manual testing were what types of characters (ASCII, alphabetic, numeric, symbols) are valid for each part of the URL, what types of values are valid (authority, port numbers, schema, etc) for each part of the URL, and what types of formatting (could some parts be omitted or placed in a different order) are permitted. As we performed these tests it became apparent to us that in our effort to produce test cases, we were breaking the URL down into each of its parts to examine more in depth what constitutes a valid scheme, authority, path, query, or fragment. This naturally lead to our next testing methodology.

Partitioning

URLValidator's isValid() method tests a URL by parsing it into its five parts -- scheme, authority, path, query, and fragments -- and making five separate calls to isValid…() functions specific to each part. This program structure along with the distinct requirements for each URL part lent itself naturally to the scheme we used to partition the input domain. The most difficult part of implementing the partition testing was developing a good set of test cases for each URL part. This was done by carefully reviewing the code for the isValid…() function that tests each partition in order to understand the rules for validation or each URL part. We created an array of ResultPairs consisting of a URL part and it's expected result for each partition. These ResultPair arrays were named as follows schemeParts, authorityParts, authorityPartsLocal, pathParts, pathPartsSlashes, queryParts, fragmentParts, fragmentPartsNO. Then we developed functions for each array which would iterate over each test case in the array, calling the appropriate isValid…() function and comparing the actual result to the expected result. The functions testing each isValid…() function were named as follows -- testIsValidScheme(), testIsValidAuthority, testIsValidPath(), testIsValidQuery(), and testIsValidFragment().

Automated Unit Testing

Lastly, we explored a more powerful testing method in order to test a larger set of test cases -- automated unit testing. To implement this testing method, we developed a function called testIsValid() that combines URL parts from five arrays, each consisting of a different URL part (created during our partition testing), in order to build complete URLs to test along with the expected results. The function uses a loop structure to accomplish this task, and generates every possible combination of the URL parts. The expected result for a URL is calculated as it is built. All parts of a URL must be valid in order for it to be valid. That is, if any part of the URL is invalid, it is an invalid URL. We test URLValidator by passing each URL generated to the isValid() function and comparing the actual result to the expected result. Utilizing this method, we were able to test 376,320 URLs.

Teamwork

When forming our group, we made an effort to ensure that we lived in the same timezone so that we might be on similar schedules. This helped to facilitate being available to work with one another. Our approach to working with one another was to establish good communication early on. We held weekly meetings, during which we would discuss the current status of the project, work together to resolve any questions or issues we were dealing with, and divide tasks and set goals for work to accomplish before the next meeting. Between meetings we would communicate via email and collaborate using Google Docs.

In our first meeting we introduced ourselves to one another, briefly reviewed Part A of the project, discussed collaboration tools, and set the goal of reviewing the URLValidator code for better understanding and completing Part A. During the week we collaborated to produce the report for Part A on Google Docs. Our second meeting consisted of discussing putting the final touches on Part A and assigning Andrew to submit. In our third meeting, we discussed manual testing individually as an exploratory exercise to become more familiar with the code. Our fourth meeting, we discussed the results of our manual testing and partitioned the code and divided test and test case development evenly. Andrew would work on the testIsValidScheme() function and test cases, Spencer would work on the testIsValidAuthority() function and test cases and testIsValidPath() function and test cases. During the we communicated with one another to make the code more consistent and had Andrew begin developing the testIsValid() function. Once all test development was completed we created a google doc for bug reporting. Each of us worked independently to test and debug the code. When a bug was found it was reported in the doc so that the others could verify it by replication and then focus on other bugs. Our last meeting consisted of us dividing the work for creating this report. All communication after was done via email and collaboration on the Google Doc used to complete the assignment.

All in all, I think our team was successful and worked well together in order to to accomplish our task.

Debugging

After finding failures in the URLValidator code with our test suites, we systematically tracked down each bug to discover the cause. Our process can, at least in part, be broken down by Agan's principles of debugging.

1. Understand the system: This was our first step before even writing the tests. In order to break something properly, we had to understand how it was intended to work and why.
2. Make it fail: This was the aim of our test cases. By testing a wide range of input to the validator in multiple combinations, we were able to identify parts of the test domain that caused the validator to behave incorrectly.
3. Divide and conquer: By narrowing test domains and working with our broader understanding of the system, we were able to isolate bugs (down to class, to function, to line).
4. Check the plug: There were times when the tests were behaving incorrectly due to incorrect testing code or other circumstantial reasons, unrelated to the URLValidator code itself. In these cases, when all else failed, addressing these issues would solve the problem and allow us to get a more accurate view of test results.
5. Get a fresh view: And, of course, collaborating on testing allowed us to bounce ideas off one another about the tests and debugging process.

Bug Reports

The bugs found during testing are as follows:

-

[URLValidator - 1]
Port numbers with greater than 3 digits return as invalid

==========

[URLValidator - 2]
Invalid IP addresses pass validator

==========

[URLValidator - 3]
Valid top-level domains are treated as invalid

==========

[URLValidator - 4]
Local URLs shown as invalid even when ALLOW_LOCAL_URLS enabled

---

-

URLValidator - 1

Port numbers with greater than 3 digits return as invalid.

Dates ----------

Created: **March 14, 2016**
Updated: **N/A**
Resolved: **N/A**

People ----------

Reported by: **Amy Dobbs, Andrew Gremmo, and Spencer Winlaw**
Email: **dobbsa@oregonstate.edu, gremmoa@oregonstate.edu, and winlaws@oregonstate.edu**

Details ----------

Type: **Bug** *Status:* **Open**
Priority: **Major** *Resolution:* **Unresolved**
Affects Version/s: **URLValidator (commit 661295fda3245c69f6a50b533faefb1f5520b550)**

Description ----------

Any complete, otherwise valid URL with a port number with greater than 3 digits will return as invalid.

Example: http://www.google.com:1234
http://www.google.com:25565

Expected Results
A valid URL with any port number within range should return as a valid URL.

Actual Results
Ports greater than 999 are shown as invalid.

Workarounds
None known.

Debugging Details
Appears to be a problem with the regular expression used to evaluate valid port numbers. This should accept any numeric string between 1 and 5 digits. Instead, it accepts any numeric string between 1 and 3 digits.
Line 158 of URLValidator.java: PORT_REGEX = "^:(\\d{1,3})$"
Fix: PORT_REGEX = "^:(\\d{1,5})$"

URLValidator - 2

Invalid IP addresses pass validator

Dates ----------

Created: **March 14, 2016**
Updated: **N/A**
Resolved: **N/A**

People ----------

Reported by: **Amy Dobbs, Andrew Gremmo, and Spencer Winlaw**
Email: **dobbsa@oregonstate.edu, gremmoa@oregonstate.edu, and winlaws@oregonstate.edu**

Details ----------

Type: **Bug** *Status:* **Open**
Priority: **Major** *Resolution:* **Unresolved**
Affects Version/s: **URLValidator (commit 661295fda3245c69f6a50b533faefb1f5520b550)**

Description ----------

The validator will pass IP addresses with a section that is greater than 255.
Example: 256.256.256.256

Expected Results
IP addresses with any section greater than 255 should be invalid.

Actual Results
IP addresses with a correct number of numeric sections pass regardless of size.

Workarounds
None known.

Debugging Details
Functions relating to validating IP addresses are in InetAddressValidator. This appears to be a bug in the function isValidInet4Address() which validates IPv4 addresses. This function checks each section for invalid numbers and characters. When checking for invalid sections greater than 255, the function returns true instead of false, allowing these to pass through the validator.

Lines 94-98 of InetAddressValidator.java:
```
if (iIpSegment > 255) {
    return true;
}
```
Fix: `return false;`

URLValidator - 3

Valid top-level domains are treated as invalid

Dates ----------

Created: **March 14, 2016**
Updated: **N/A**
Resolved: **N/A**

People ----------

Reported by: **Amy Dobbs, Andrew Gremmo, and Spencer Winlaw**
Email: **dobbsa@oregonstate.edu, gremmoa@oregonstate.edu, and winlaws@oregonstate.edu**

Details ----------

Type: **Bug** *Status:* **Open**
Priority: **Major** *Resolution:* **Unresolved**
Affects Version/s: **URLValidator (commit 661295fda3245c69f6a50b533faefb1f5520b550)**

Description ----------

Valid Country Code top-level domains that do not pass the validator include:
.na
.za

Expected Results
URLs which include these valid top-level domains should pass the validator.

Actual Results
The validator shows URLs including these top-level domains to be invalid.

Workarounds
None known.

Debugging Details
DomainValidator.java contains a list of local top-level domains, COUNTRY_CODE_TLD_LIST. The TLDs of URLs are compared to this list as part of validity checks. The list is incomplete and should include .je - .zw of the lis which can be found here https://en.wikipedia.org/wiki/Country_code_top-level_domain.

Line 357-359 of DomainValidator.java: "it", // Italy
};
Fix: “je”, …, “zw};

URLValidator - 4

Local URLs shown as invalid even when ALLOW_LOCAL_URLS enabled

Dates ----------

Created: **March 14, 2016**
Updated: **N/A**
Resolved: **N/A**

People ----------

Reported by: **Amy Dobbs, Andrew Gremmo, and Spencer Winlaw**
Email: dobbsa@oregonstate.edu, gremmoa@oregonstate.edu, and winlaws@oregonstate.edu

Details ----------

Type: **Bug** *Status:* **Open**
Priority: **Major** *Resolution:* **Unresolved**
Affects Version/s: **URLValidator (commit 661295fda3245c69f6a50b533faefb1f5520b550)**

Description ----------

URLs such as localhost and localdomain show as invalid even when the validator is set to allow local URLs.

Example: localhost
ftp://localdomain/

Expected Results

Local URLs should pass the validator when the ALLOW_LOCAL_URLS option is enabled.

Actual Results

Local URLs always fail the validator even when ALLOW_LOCAL_URLS is enabled.

Workarounds

None known.

Debugging Details

Local URLs are evaluated in DomainValidator in the isValidLocalTld() function. They are compared to a list of commonly used local domains (i.e. localhost and localdomain) and should return true if the domain is contained in the list. Instead, the inverse is returned.

Line 204 of DomainValidator.java: return !LOCAL_TLD_LIST.contains(...);
Fix: return LOCAL_TLD_LIST.contains(...);