Final Project: UrlValidator - Part B
Class: CS362 Software Engineering II, Winter 2016
Project Team: Janel Buckingham (buckinja), Jeannine Amm (ammj), Paul Zotz (zotzp)

Sections:

Methodology of Testing

Our methodology of testing was to build a good input partition that tested valid input, invalid input, boundary conditions and provided good code coverage for UrlValidator.java and DomainValidator.java.

For measuring the code coverage we utilized the tool eclemma[1] in eclipse with the following results for branch coverage:

DomainValidator.java - 99.1%
UrlValidator.java - 95%

Testing more of the long options when setting up URLValidator would have improved the coverage in our target files by a marginal amount. 100% coverage would not have been achievable as a number of the missed branches appeared to be due to unreachable code. Two examples are provided.

Example 1: isValidFragment regex accepts any character and while there may be a character it does not accept, we were unable to identify one.
Example 2: if (extra != null && extra.trim().length() > 0)
Of the following two branches only one would ever be valid in all situations:

extra == null && extra.trim().length() > 0
extra == null && extra.trim().length() < 0

Manual Testing

Tests: testManualTest1(), testManualTest2()
Regular Urls Examples

- htps://www.google.com/
- https://google.com/something.php
- https://google.com/something.php
- https://altavista.com
- http://altavista.d/

Complex Urls Examples

- https://mail.google.com/mail/u/0/#inbox
- https://oregonstate.instructure.com/courses/1568425/assignments/6656123?module_item_id=16591766
- https://oregonstate.instructure.com/courses//assignments/6656123?module_item_id=16591766
- https//oregonstate.instructure.com/courses/
- https://oregonstate.instructure/courses
- http://oregonstate.instructure.com/courses/1568425/
- http://localhost:8080/_ah/api/explorer
- http://google.de/index.html>file
- 3tp://www.altavista.com
- http://www.mydomain.com\test
- ftps://www.mydomain.com/|value/file
- http://altavista.com/index.php?iam=aparam
- ftp://user@test.com:80
- ftp://user:password@test.com:80
- http://www.altavista.comwww.altavista.com
- http://www

Boundary Condition Examples

- null
- \u1D200 // non-ascii character

Input Partitioning:

Tests: testTWayInputPartitionLocal(), testTWayInputPartitionAll();
For the input partitioning portion we split a URL into scheme, post scheme, hier-part, host, tld, port, path, query and fragment.

For each part of the urls we identified one or two components of the url that would pass, one or two components of the url that would fail, a zero length string that would either pass or fail depending on the part of the url it fell under and null.

Using the Combinatorial Tool jenny[2], a set of 5-way combinatorial input indexes was generated. 5-way was chosen for a the following reasons:

- it was the highest number of combinatorial inputs that produced a set of inputs in a reasonable time frame.
- “67% of failures were triggered by only a single parameter value”[3]
- “93% by 2-way combinations”[3], but we found when failure entries were also included during our test 2-way resulted in minimal bug identification.
- “100% detection with 4 to 6 way interactions”[3]
- As a URL has infinite possible combinations, we could not reasonably achieve mapping of 100% of the possible valid inputs. To get a good representation we chose to map portions with both valid and invalid partition.
- If we had chosen to iterate through every possible combination of even the small subset we would have had to test:
  5 * 5 * 4 * 4 * 6 * 12 * 5 * 6 * 4 * 5 = 17280000 Urls total.

The result of the 5Way Combinatorial testing input yielded 17590 Urls for testing. This number is considerably less than 17280000 but tests every combination of 5 parts and provided great potential for finding a large number of bugs.

Challenges:
17590 still proved to be a large enough number to exceed the maximum size of a java file in eclipse (65536 bytes) when indexes were generated from a data structure inside the file. We chose to put the indexes into a external file called twayIndexes.txt and build the urls and truth values from the indexes to alleviate this constraint.

Calculating the Pass/Fail for the full URL:
The expected result was calculated based on the part expected results using the following:

- if the host was an ip and there was no TLD extension the hostname expected value applied to both ip and TLD
- if the host was false and a TLD was being attached, the host.tld was checked if valid and host set to true if valid.
- otherwise all expected values were AND’ed together (&&) to produce the expected result for the url.

Note: Jenny was built using an older compiler and needed to be edited to update a number of the outputs. These edits mapped %d to %ld or %lu.

Input into Jenny: ./jenny -n5 5 4 4 6 12 5 6 4 5

The generated combinations were stored in the file twayIndexes.txt and read by the program from that file.

Unit Tests / Random Tests:

We decided to divide the unit tests as follows in JUnit: testIsValidScheme(), testIsValidAuthority(), testIsValidPath(), testIsValidQuery(), testIsValidFragment(), testTWayPartitionLocal() (isValid()), testTWayPartitionAll() (isValid())

Each unit test was given a data structure of valid and invalid inputs to test and verify that the outcome was as expected. We were able to run the tests as a suite to see all results at once and individually to focus on the bugs that were found.

How did we work in the team?

We worked cohesively together in our team environment. We all brought different strengths to the table which benefitted the project. We all acted professionally and met our deliverables as agreed.

How did we divide our work?

We first did a requirements analysis to understand what the project was asking us and to make sure we were all on the same page. We charted out how we wanted to test the system and came up with a couple of partition strategies. We estimated the work effort that each person could reasonably complete in the next few weeks based on available time. We picked the strategy that we felt satisfied the project deliverables and could be completed considering the time restraints.

To ensure we all benefitted from the programming, testing and debugging experience we chose to all participate in each portion of the project and split each section.

We each took a portion of the manual testing, as it was not enough work for one person, and split the input partitioning implementation and automated testing implementation into 3 units, each taking 1 unit.

We ran the test and came up with 7 bugs found. We split the bugs into 3 sections and each debugged and documented a portion of the bugs.

How did we collaborate?

We used several collaboration tools to complete the project:

- Github for storing our tests and code
    - https://github.com/jamm8888/CS362Project
- Google docs for sharing our documents
- Google hangouts for discussions
- Pastebin.com for sharing quick code snippets

Bug Reports:

URLVALIDATOR-001
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:

What is the failure?
Authorities containing a number of country top-level domains are not accepted as valid. Authorities containing top-level domains starting with .ac (Ascension Island) and continuing through .it (Italy) validate correctly but valid country codes listed on Wikipedia[2] starting with .je (Jersey) and continuing alphabetically through .zw (Zimbabwe) do not validate as they should.

How did you find it?
Unit test for isValidAuthority() - testing each country TLD and comparing return value with expected value.

What is the cause of that failure? Explain what part of code is causing it?
String array COUNTRY_CODE_TLDS in DomainValidator.java (starting at line 248) is missing all expected values for country code top-level domains after ".it"; remaining country codes from .je to .zw should be added to this array.

1. https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains#Country_code_top-level_domains

URLVALIDATOR-002
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines

Labels: None

Description:

What is the failure?
Port numbers do not validate correctly for the entire range of valid ports. Ports consisting of three or fewer digits (80, 800, etc.) validate as expected but valid ports[2] consisting of four or five digits (8080, 9000, 30000, 65535) are incorrectly treated as invalid.

How did you find it?
Unit test for isValidAuthority() - testing valid ports from 1 to 5 digits.

What is the cause of that failure? Explain what part of code is causing it?
Error in string PORT_REGEX at line 158 of UrlValidator.java. Value is "^:(\\d{1,3})$", should be "^:(\\d{1,5})$" to allow ports of up to 5 digits.

1. https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

URLVALIDATOR-003
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:

What is the failure?
All non-null query strings are validated incorrectly. Invalid query values such as newline characters are treated as valid while valid queries in the “key=value” format[2] are treated as invalid. For any given non-null query value, the validator treats its as having the opposite of its expected validity.

How did you find it?
Unit test for isValidQuery() - testing valid ports from 1 to 5 digits.

What is the cause of that failure? Explain what part of code is causing it?

In function isValidQuery() - return statement at line 446 should not have exclamation point before return value. This causes the function to always return the opposite of the expected value unless the branch for a null query string is taken above. Removing the exclamation point should restore expected behavior.

1. https://tools.ietf.org/html/rfc3986#section-3.4

URLVALIDATOR-004
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:

What is the failure?
manualurl2 Fail:http://localhost/
expected: true result: false
manualurl2 Fail:http://localhost:8080/_ah/api/explorer
expected: true result: false

For both of these tests, the ALLOW_LOCAL_URLS flag has been set in the urlValidator constructor, but the localhost addresses are not passing as valid urls.

How did you find it?
Manual url test - testManualTest2()

What is the cause of that failure? Explain what part of code is causing it?
The bug is located in DomainValidator. On line 139, the

```
if (!hostnameRegex.isValid(domain)){...}
```
//where domain = “localhost”

condition is not met because the valid input of “localhost” returns true and is being negated by the “!” symbol. The “!” symbol seems to be there by mistake. Because this if statement is not entered, the isValid function returns false.

URLVALIDATOR-005
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:
URLValidator returns false for url in the format <schema>://<user>@<host>:<port>

How did you find it?
Manual url test - testManualTest1()
new Url("ftp://user@test.com:80",true)

What is the cause of that failure?

File: UrlValidator.java
Lines: 381 to 389 checks for valid host
if (!domainValidator.isValid(hostLocation))
Line 135 in DomainValidator.java
function isValid(hostname)
checks the hostname against the regex:
RegexValidator{^(?:\p{Alnum}(?>[\p{Alnum}-]*\p{Alnum})*\.)+(\p{Alpha}{2,})$}

The regex doesn't appear to allow for "@" in the pattern before the first "." and returns false.
Note: In Apache's bug reports this also maps to VALIDATOR-387[2]

1. https://issues.apache.org/jira/browse/VALIDATOR-387

URLVALIDATOR-006
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:
URLValidator returns false for url in the format <schema>://<user>:<password>@<host>:<port>
This is classified as major as any url checker verifying ftp user information in this format will reject the url.

How did you find it?
Manual url test - testManualTest1()
new Url("ftp://user:password@test.com:80",true),

File: UrlValidator.java
Lines: 381 to 389 checks for valid host
if (!domainValidator.isValid(hostLocation))
Line 135 in DomainValidator.java
function isValid(hostname)
checks the hostname against the regex:
RegexValidator{^(?:\p{Alnum}(?>[\p{Alnum}-]*\p{Alnum})*\.)+(\p{Alpha}{2,})$}

The regex doesn't appear to allow for "@" or ":" in the pattern before the first "." and returns false.

Note: In Apache's bug reports this also maps to VALIDATOR-353[2] which was fixed in the apache version 1.5.0
Apache's fix was to change the regex to:
"^(?:\\p{Alnum}(?>[\\p{Alnum}-]{0,61}\\p{Alnum})?\\.)+(\\p{Alpha}(?>[\\p{Alnum}-]{0,61}\\p{Alnum})?)\\.?$"[2]

1. https://issues.apache.org/jira/browse/VALIDATOR-353

URLVALIDATOR-007
Details:
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved
Affects Version(s):
Fix Version/s: None
Component/s: Routines
Labels: None

Description:

What is the failure?

twayPartition1 Fail: ftp://192.168.0.1.localhost:8080/path/to/something?ima=aquery
expected: true result: false

For this test, the ALLOW_LOCAL_URLS flag has been set in the urlValidator constructor, but the localhost tlds are not passing as valid urls.

How did you find it?
input partition / isValid unit test - twayPartition1()
ftp://192.168.0.1.localhost:8080/path/to/something?ima=aquery

What is the cause of that failure? Explain what part of code is causing it?
The bug is located in DomainValidator. On line 204, the

return !LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));

condition is not met because the valid input of tld “localhost” returns true and is being negated by the “!” symbol and then returned. The “!” symbol seems to be there by mistake.

Debugging and Agans’ Principles:

We made use of all of Agans’ Principles[4] during our debugging phase.

Rule #1: Understand the system
Initially we stepped through the program to understand the system. It also helped to reference the RFC standards when determining what a valid or invalid url should consist of.

Rule #2: Make it fail
We were successfully able to make the system fail. Once the system failed with the manual tests we were able to understand why and come up with good input partitioning to make the system fail again. Those tests gave us more insight to devise input partitioning for the unit testing that would make the system fail yet again during unit testing.

Rule #3: Quit Thinking and look
This rule was easier on the manual tests than it was on the input partitioning. The manual tests contained less data and were a good source for starting to identify the fixes for the bugs. Clearing some of the manual bugs made it easier to see what the data was telling us in the input partitioning and unit tests.

Rule #4: Divide and Conquer

While we did not utilize delta debugging we used debugging tools that allowed us to set breakpoints, step through the program and view the variable values to narrow down the problem. (e.g. Eclipse debugger).

Rule #5: Change One thing at a time
This rule was followed quite a bit as there were quite a number of errors. Fixing one thing at a time made the list smaller and more manageable.

Rule #6: Keep an audit trail
As these were small bug fixes we did not keep an audit trail of what steps had been performed during debugging. We did, however, keep a rolling conversation audit trail of where we were when writing the testing.

Rule #7: Check the plug
After we finished our tests we verified the tests against UrlVaidatorCorrect code to check the plug. If the test failed on the correct version, eclipse debugger was used to step through and verify if it was the test code causing the problem.

Rule #8: Get a Fresh View
Many instances in the process we passed our code back and forth and each ran the tests. This helped us find issues we may not have seen otherwise.

Rule #9: if you didn't fix it, it ain't fixed.
A final run was performed of the tests with all the fixes in place and the tests passed.

[1] EclEmma, Java Code Coverage for Eclipse, http://eclemma.org
[2] http://burtleburtle.net/bob/math/jenny.html
[3] Kuhn, D. R., Kacker, R. N., Lei, Y., *Practical Combinatorial Testing*, ch. 6, National Institute of Standards and Technology, Oct. 2010, http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf
[4] https://oregonstate.instructure.com/courses/1568425/files/63553192/download?wrap=1