Term Project

URLValidator

Part I

CS362 - Winter 2016

Lucia Blackwell
Alicia Broederdorf
David Rigert
Karen Thrasher

Part I - URLValidator Test Plan

1 Introduction

Apache URLValidator is a Java library that facilitates validation of commonly-used Web address formats.

1.1 Current Test Coverage

The Apache URLValidator project as provided does not currently have any test coverage. Thus, all testing added will be an improvement to the current state.

2 Features to Test

The URLValidator library primarily includes four features:

1. DomainValidator
2. InetAddressValidator
3. RegexValidator
4. UrlValidator

Improvements to the current testing state for all of these features will include mutation testing and code coverage using Pitest and/or Cobertura, as well as static analysis using FindBugs.

3 Approach

FindBugs will be used to apply static analysis to the entire code base.

3.1 DomainValidator

The `DomainValidator` class verifies that a string is a valid Internet domain name, according to IETF standards. `isValid()` is the primary public function in `DomainValidator`, and verifies that a string is a valid domain name. `DomainValidator` also has functions which check that strings are valid top-level domains of several varieties.

3.1.1 - Input Domain Partitioning (IDP)

Each string-validating function of `DomainValidator` will be given strings containing a known valid and known invalid domain, and the results verified.

3.1.2 - Mutation Testing

Pitest will be used to conduct mutation testing on `DomainValidator` in order to check whether test results are affected by code changes.

3.1.3 - Code Coverage

Code coverage of `DomainValidator` will be analyzed using Cobertura or Pitest.

3.2 InetAddressValidator

The `InetAddressValidator` class verifies that a string is a valid IP address. The class contains two main methods, `isValid()`, which checks to confirm that a string is a valid IP address, and `isValidInet4Address()`, which verifies that an address is a valid IPv4 address.

3.2.1 - Input Domain Partitioning (IDP)

IDP testing will consist of providing valid IP addresses and invalid strings to confirm if the methods are able to detect if a valid IP address was passed in.

3.2.2 - Mutation Testing

Mutation testing using Pitest will be used to determine if the test suite is able to detect changes in the code and provide appropriate results.

3.2.3 - Code Coverage

Code coverage will be analyzed using either Cobertura or Pitest .

3.3 RegexValidator

The `RegexValidator` class verifies a value against a set of regular expressions. The class contains four constructors to allow either a single or set of regular expressions to be specified, as well as indicating if the validation should be case sensitive. `RegexValidator` contains four methods which return if a string is valid, the groups of regular expressions the value matches, and the validator's regular expressions.

3.3.1 - Input Domain Partitioning (IDP)

IDP testing will consist of providing a set of regular expression along with value strings that both match and do not. These domains will further be divided into case sensitive and insensitive.

3.3.2 - Mutation Testing

Pitest will be used to conduct mutation testing to ensure the suite of unit tests are able to detect changes in the code, verified by the appropriate responses being returned.

3.3.3 - Code Coverage

Code coverage will be analyzed using either Cobertura or Pitest .

3.4 UrlValidator

The `UrlValidator` class verifies that a URL string has valid syntax based on RFC 2396, including the scheme and the address. The class contains only one public method, `isValid()`, which accepts a `String` object containing the URL to verify, and numerous constructor overloads to configure the validation logic. The class uses `DomainValidator` and `RegexValidator` internally to verify parts of the URL.

3.4.1 - Input Domain Partitioning (IDP)

IDP will be used with a functionality-based approach to divide the inputs into disjoint domains and create unit tests to verify that the validator correctly validates a representative value from each domain, and with each combination of options.

3.4.2 - Random Testing

Random testing will be used to generate random URL strings that comply with RFC 2396 and verify that the `UrlValidator` object is capable of correctly handling all variations.

3.4.3 - Code Coverage

Cobertura or Pitest will be used to verify sufficient statement coverage of the unit tests.

3.4.4 - Mocking

Mock objects will be used to verify the logic of the `isValid()` method independent of the implementations of the `DomainValidator` and `RegexValidator` classes.

4 New Tools

The tests will utilize two new tools, both FindBugs to perform static analysis as well as Pitest. Pitest provides code coverage, which previously has been accomplished using Cobertura, but has the added benefit of completing mutation testing as well.