PIT Mutation Tool Additions

https://github.com/matthewbachelder/cs6367.git

Matthew Bachelder
University of Texas at Dallas
United States
mtb140130@utdallas.edu

Vaishnavi Bhosale
University of Texas at Dallas
United States
vbb160030@utdallas.edu

Richard Fisher
University of Texas at Dallas
United States
rxf120030@utdallas.edu

ABSTRACT

Mutation testing is a method of software quality control that tests the unit test suite defined for programs by artificially injecting errors, or mutations, into software. Tests are then executed to evaluate the quality of the test suite. If the mutations are detected, then the mutation is killed and the test suite is deemed adequate with respect to that section of code. If the mutation is not detected, then it is said to have passed, and the test suite for that section of code is inadequate.

CCS CONCEPTS

• **Software engineering** → **Software testing and debugging**;

KEYWORDS

PIT Mutation testing, Mutators, Operators

1 INTRODUCTION

Mutation testing is a method of software quality control that tests the unit test suite defined for programs by artificially injecting errors, or mutations, into software. Tests are then executed to evaluate the quality of the test suite. If the mutations are detected, then the mutation is killed and the test suite is deemed adequate with respect to that section of code. If the mutation is not detected, then it is said to have passed, and the test suite for that section of code is inadequate.

Mutations, or faults that are injected into code, are modifications to various operators within the code. There are numerous mutations that can be used but commonly used examples include:

- Modification of increment operators to decrement
- Negation of certain values
- Modification of constants
- Modification of comparison operators

2 Proposed Additions to PIT

This project will add several mutations to the suite of available code modifications. These include:

a. ABS: Replaces a variable by its negation, e.g., a becomes –a
b. OBBN: Replaces the operators & by | and vice versa, e.g., a&b becomes a|b
c. AOD: Replaces an arithmetic expression by one of the operand, e.g., a + b becomes a
d. ROR: Replaces the relational operators with another one. It applies every replacement, e.g., < becomes ≥, or > becomes ≤
e. AOR: Replaces an arithmetic expression by another one. a + b becomes a ∗ b
f. UOI: Replaces a variable with a unary operator or removes an instance of an unary operator. a becomes a++
g. CRCR: Replaces a constant a with its negation, or with 1, 0, a + 1, a – 1, e.g., a becomes –a, and a becomes a – 1.

3 PIT: REAL WORLD MUTATION

PIT is a mutation testing framework for Java developed to support the day to day development on real codebases. This means that PIT aims at: (1) having a good integration with build tools like Maven, Ant, Gradle and integrated development environments like Eclipse or IntelliJ. (2) being fast: PIT uses three techniques to obtain its results: working on bytecode instead of source code, selecting the tests to run against the mutants and minimizing the number of mutant executions. (3) making a clear report of the tests execution. This makes the navigation between source code and mutants easy by highlighting mutants that were not killed.

4 Experimental Evaluation

The proposed modifications to PIT will be evaluated on five real world projects chosen from GitHub. Each project will contain a minimum of 1,000 lines of code, and will be evaluated after running at least 50 tests per project. The overall quality of the test suites, when tested with the augmented PIT, will be measured using the mutation adequacy score given in equation (1).

$$MAS(P,TS) = \frac{Kn}{M-En} \quad (1)$$

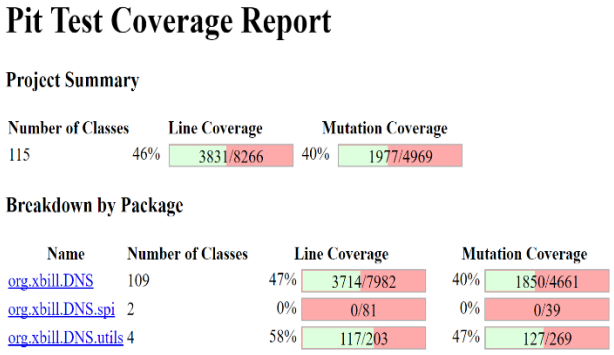
Where, P = Program under test, TS = Test suite, Kn = number of Killed mutants, M = total number of Mutants, En = number of equivalent mutants.

5 Results and Discussion

5.1 dnsjava Test Project

To test the augmentation of mutators we used different projects. Dnsjava is one of the project to test the code. The aim of dnsjava project is to have straightforward way to build and deploy arbitrary versions of the dnsjava library as maven artigacts and make them available as maven managed dependencies.

This is a pre-augmentation report of dnsjava project.



Pit Test Coverage Report

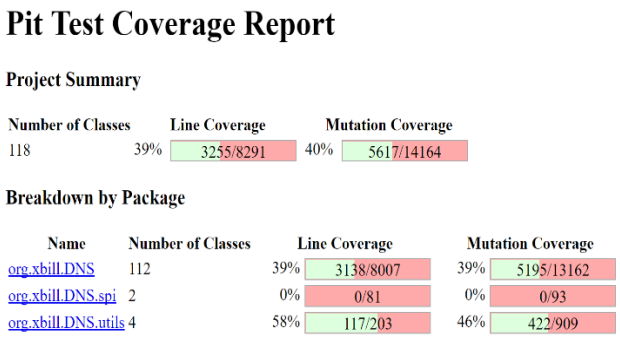
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
115	46%	3831/8266	40%	1977/4969

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.xbill.DNS	109	47%	3714/7982	40%	1850/4661
org.xbill.DNS.spi	2	0%	0/81	0%	0/39
org.xbill.DNS.utils	4	58%	117/203	47%	127/269

Following is a post-augmentation report of dnsjava report.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
118	39%	3255/8291	40%	5617/14164

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.xbill.DNS	112	39%	3138/8007	39%	5195/13162
org.xbill.DNS.spi	2	0%	0/81	0%	0/93
org.xbill.DNS.utils	4	58%	117/203	46%	422/909

Line Coverage: This indicates the line coverage for the tests. In this case, the tests cover 39% of the code which is dropped from 46%. Higher percentage is better like 100% indicates every line of code is tested.

Mutation Coverage: This indicates the performance of the mutated tests. The number is killed/total, indicating how many mutations were killed. Higher percentage better like 100% indicates all mutations were killed.

In case of mutation coverage, 5617 mutators are killed after augmentation. There are total 13162 mutators after augmenting new mutators. From the values shown in the report we can calculate mutation adequacy score which is 0.3965.

There are three packages in the dnsjava project as shown in the figure. These packages can be separately examined to see the test coverage report. Following is the org.xbill.DNS package summary before augmentation.

Pit Test Coverage Report

Package Summary

org.xbill.DNS

Number of Classes	Line Coverage		Mutation Coverage	
109	47%	3714/7982	40%	1850/4661

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
A6Record.java	100%	53/53	91%	31/34
AAAARecord.java	71%	22/31	47%	9/19
AFSDBRecord.java	100%	6/6	100%	3/3
APLRecord.java	96%	142/148	84%	69/82
ARecord.java	93%	25/27	96%	27/28
Address.java	85%	155/183	80%	149/187
CAARecord.java	3%	1/35	0%	0/8
CERTRecord.java	2%	1/63	0%	0/18
CNAMERecord.java	100%	6/6	100%	3/3
Cache.java	48%	183/379	6%	15/239
Client.java	73%	24/33	0%	0/18

Following is post-augmented report of org.xbill.DNS package.

Pit Test Coverage Report

Package Summary

org.xbill.DNS

Number of Classes	Line Coverage		Mutation Coverage	
112	39%	3138/8007	39%	5195/13162

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
A6Record.java	100%	53/53	91%	81/89
AAAARecord.java	71%	22/31	33%	17/51
AFSDBRecord.java	100%	6/6	100%	8/8
APLRecord.java	96%	142/148	84%	248/294
ARecord.java	93%	25/27	96%	76/79
Address.java	61%	112/183	68%	409/599
CAARecord.java	3%	1/35	0%	0/17
CERTRecord.java	2%	1/63	0%	0/34
CNAMERecord.java	100%	6/6	100%	6/6
Cache.java	0%	0/379	0%	0/746
Client.java	0%	0/33	0%	0/39

From the above diagram, we can say that line coverage and mutation coverage are dropped. More number of classes are tested from a package after augmenting.

5.2 corenlp project

Stanford CoreNLP provides a set of natural language analysis tools written in java. It can take raw human language text input and give the base forms of words and we used this to analyze the addition of new mutators.

This is the pre-augmentation testing report of the corenlp project.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
129	28%	3870/13727	24%	2535/10753

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
edu.stanford.nlp.math	5	19%	261/1406	17%	275/1662
edu.stanford.nlp.util	99	33%	3295/10123	29%	2182/7655
edu.stanford.nlp.util.concurrent	6	26%	120/469	17%	45/268
edu.stanford.nlp.util.logging	19	11%	194/1729	3%	33/1168

Report generated by PIT 1.2.4

After augmentation, the test coverage report of corenlp project is as shown below.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
129	28%	3889/13769	22%	6869/30943

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
edu.stanford.nlp.math	5	19%	261/1406	15%	1013/6592
edu.stanford.nlp.util	99	33%	3306/10141	27%	5674/20794
edu.stanford.nlp.util.concurrent	6	26%	120/469	15%	105/709
edu.stanford.nlp.util.logging	19	12%	202/1753	3%	77/2848

Report generated by PIT 1.2.4

We can conclude that after adding seven new mutators the mutation coverage lower down by 2%.

Corenlp is project is a big project among all other projcts on which we tested the augmentation. It requires a lot of time almost 24 hours to run the project. The solution to this problem is to use concurrent programming. Threads are created to run the independent packages simultaneously. This way we can get the output early. In above scenario, we only used some packages from NLP, and not the entire project itself. In the images we can see the packages selected to test the augmentation.

5.3 jsoup project

jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery like methods. Following is the pre-augmentation code coverage report of jsoup.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
54	81%	5809/7196	66%	3162/4788

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.jsoup	5	63%	30/48	70%	16/23
org.jsoup.examples	3	0%	0/100	0%	0/66
org.jsoup.helper	6	81%	748/921	78%	462/592
org.jsoup.internal	2	85%	44/52	88%	42/48
org.jsoup.nodes	14	87%	1198/1376	72%	687/948
org.jsoup.parser	14	77%	2742/3572	59%	1435/2438
org.jsoup.safety	2	95%	275/290	66%	106/161
org.jsoup.select	8	92%	772/837	81%	414/512

After augmenting the mutators the test coverage report will be look like below

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
55	81%	5837/7224	69%	7359/10632

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.jsoup	5	63%	30/48	58%	18/31
org.jsoup.examples	3	0%	0/100	0%	0/195
org.jsoup.helper	6	81%	753/926	79%	962/1221
org.jsoup.internal	2	85%	44/52	94%	146/156
org.jsoup.nodes	14	87%	1201/1379	77%	1576/2056
org.jsoup.parser	14	77%	2749/3579	64%	3400/5331
org.jsoup.safety	2	95%	279/294	76%	370/490
org.jsoup.select	9	92%	781/846	77%	887/1152

In this test suite, line coverage is 81% means it covered a lot of the code. Mutation coverage is increased after augmentation.

5.4 twilio

Twilio is another project used to test the code coverage report. Twilio java uses Maven. At present the jars are available from a public maven repository. Before augmentation, the report is shown below.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1086	47%	23152/49331	34%	6978/20322

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
com.twilio	1	70%	31/44	29%	6/21
com.twilio.base	7	52%	96/185	20%	21/103
com.twilio.converter	5	81%	46/57	87%	26/30
com.twilio.example	3	0%	0/107	0%	0/24
com.twilio.example.resource	5	0%	0/69	0%	0/29
com.twilio.exception	3	43%	16/37	44%	4/9
com.twilio.http	8	72%	208/289	47%	55/116
com.twilio.jwt	1	100%	31/31	90%	9/10
com.twilio.jwt.accesstoken	8	78%	106/136	64%	45/70
com.twilio.jwt.client	4	96%	97/101	96%	24/25
com.twilio.jwt.taskrouter	5	87%	125/144	82%	47/57
com.twilio.jwt.validation	2	98%	156/160	96%	43/45

After augmenting mutation is lowered by 7%. There are 12 packages in this project. We can analyze individual packages in PIT. It generates test coverage report for all packages and all java files in the package can be examined. Post-augmentation report is as shown below

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1087	47%	23155/49334	27%	8023/29753

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
com.twilio	1	70%	31/44	25%	6/24
com.twilio.base	7	52%	96/185	19%	27/142
com.twilio.converter	5	81%	46/57	91%	41/45
com.twilio.example	3	0%	0/107	0%	0/36
com.twilio.example.resource	5	0%	0/69	0%	0/39
com.twilio.exception	3	43%	16/37	36%	4/11
com.twilio.http	8	72%	208/289	38%	93/243
com.twilio.jwt	1	100%	31/31	91%	10/11
com.twilio.jwt.accesstoken	8	78%	106/136	60%	51/85
com.twilio.jwt.client	4	96%	97/101	81%	51/63
com.twilio.jwt.taskrouter	5	87%	125/144	69%	110/159
com.twilio.jwt.validation	2	98%	156/160	86%	83/97

5.5 webmagic

Webmagic is a scalable crawler framework. It covers the whole lifecycle of crawler like downloading, url management, content extraction. It can simplify the development of a specific crawler. The review of the code coverage report before augmentation is shown below.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
50	54%	830/1542	35%	313/896

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
us.codecraft.webmagic	5	42%	212/503	18%	62/350
us.codecraft.webmagic.downloader	6	84%	181/215	67%	73/109
us.codecraft.webmagic.model	1	40%	12/30	38%	3/8
us.codecraft.webmagic.pipeline	3	31%	11/36	0%	0/14
us.codecraft.webmagic.processor	1	0%	0/11	0%	0/5
us.codecraft.webmagic.processor.example	3	21%	10/47	9%	2/22
us.codecraft.webmagic.proxy	2	72%	33/46	36%	16/45
us.codecraft.webmagic.scheduler	3	84%	37/44	64%	18/28
us.codecraft.webmagic.scheduler.component	1	57%	4/7	60%	3/5
us.codecraft.webmagic.selector	17	58%	249/433	48%	109/228
us.codecraft.webmagic.thread	1	18%	7/40	0%	0/19
us.codecraft.webmagic.utils	7	57%	74/130	43%	27/63

After augmenting, the test coverage report is as shown below.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
50	54%	830/1542	29%	491/1675

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
us.codecraft.webmagic	5	42%	212/503	12%	79/654
us.codecraft.webmagic.downloader	6	84%	181/215	64%	90/141
us.codecraft.webmagic.model	1	40%	12/30	44%	4/9
us.codecraft.webmagic.pipeline	3	31%	11/36	0%	0/18
us.codecraft.webmagic.processor	1	0%	0/11	0%	0/5
us.codecraft.webmagic.processor.example	3	21%	10/47	4%	2/49
us.codecraft.webmagic.proxy	2	72%	33/46	34%	42/123
us.codecraft.webmagic.scheduler	3	84%	37/44	64%	25/39
us.codecraft.webmagic.scheduler.component	1	57%	4/7	50%	4/8
us.codecraft.webmagic.selector	17	58%	249/433	40%	185/457
us.codecraft.webmagic.thread	1	18%	7/40	0%	0/27
us.codecraft.webmagic.utils	7	57%	74/130	41%	60/145

Summary

Before augmentation test suit summary is given below.

	Code Coverage	Muation coverage	MAS(P,TS)
Dnsjava	46%	40%	.3978
Jsoup	81%	66%	.6604
Twilio	41%	34%	.3433
Webmagic	54%	35%	.3493
Corenlp	28%	24%	.2402

After augmentation the test suite summary is given below.

	Code Coverage	Muation coverage	MAS(P,TS)
Dnsjava	39%	40%	.3965
Jsoup	81%	69%	.6921
Twilio	47%	27%	.2696
Webmagic	54%	29%	.2931
Corenlp	28%	22%	.2194

Conclusion

PIT offers significant potential for evaluating a test suite but its test suite is incomplete. The availability of the source code for the mutators allows one to extend the existing mutations, either as a general improvement to the core suite of mutators, or as extensions to meet the specific needs of a project or portfolio. PIT also offers significant reporting capabilities, allowing precise identification of inadequacies in a test suite.

The existing state of test suites is poor. Although they could be outliers, five projects were evaluated and only one had a mutation score (MAS) above 0.50. The feedback that PIT can provide the necessary feedback to a team to improve their test suite.

A HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are deferent in the appendices. In the appendix environment, the command section is used to indicate the start of each Appendix, with alphabetic order designation (i.e., the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure within an Appendix, start with subsection as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

REFERENCES

[1] [11] H. Coles. 2017. “PIT.” Retrieved from http://pitest.org/.

[2] Coles, Henry, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. "PIT: a practical mutation testing tool for Java." In Proceedings of the 25th International Symposium on Software Testing and Analysis, pp. 449-452. ACM, 2016.

[3] Laurent, Thomas, Mike Papadakis, Marinos Kintis, Christopher Henard, Yves Le Traon, and Anthony Ventresque. 2017. Assessing and Improving the Mutation Testing Practice of PIT. 430-435. 10.1109/ICST.2017.47.

[4] Rani, Shweta, et al. “Experimental Comparison of Automated Mutation Testing Tools for Java.” *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2015, doi:10.1109/icrito.2015.7359265.