Is Mutation Score a Fair Metric?

Beatriz Souza
Federal University of Campina Grande, Campina Grande
Brazil
beatriz.souza@ccc.ufcg.edu.br

Abstract

Comparing the mutation scores achieved for test suites, one is able to judge which test suite is more effective. However, it is not known if the mutation score is a fair metric to do such comparison. In this paper, we present an empirical study, which compares developer-written and automatically generated test suites in terms of mutation score and in relation to the detection ratios of 7 mutation types. Our results indicate fairness on the mutation score.

CCS Concepts **• Software and its engineering → Software testing and debugging**;

Keywords Mutation Testing, Mutation Score, Test Suite Effectiveness

ACM Reference Format:
Beatriz Souza. 2019. Is Mutation Score a Fair Metric?. In *Proceedings of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '19), October 20–25, 2019, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3359061.3361084

1 Research Problem and Motivation

Mutation Testing [5, 9, 10] is a fault-based testing technique where syntactic changes are inserted in the original program, by the application of mutation operators, to create faulty programs called mutants. Mutants caught by a test suite are said to be killed. One outcome of the Mutation Testing process is the mutation score [10], which represents the number of killed mutants and indicates the ability of a test suite to detect faults.

Suppose we apply Mutation Testing to measure the effectiveness of test suites produced by two different unit test generation methods, A and B. By doing so, we compare the mutation scores of A and B test suites.

To simplify the problem, suppose we applied just two mutation operators, MO1 and MO2 and MO1 creates 60% of all mutants. If A's test suites detect all mutants of type MO1, but no mutant of type MO2 and B's test suites detect all mutants of type MO2, but no mutant of type MO1, A's test suites may have a higher mutation score. But this may not indicate that A is better than B regarding mutation detection as in fact their test suites detect different types of mutation.

In this paper, we compare developer-manually written (MTSs) and automatically generated test suites (ATSs). We use a set of 10 open-source projects and a set of 7 mutation operators. Our goal is to answer the following research questions: [RQ1] Which test generation method produces test suites that detect more mutants according to each mutation type?, and [RQ2] Is it fair to compare test suites quality using the mutation score?

2 Related Work

Previous studies have investigated the use of mutation testing for software testing experiments [5, 9, 11]. But they mainly investigate whether mutants are a valid substitute for real faults. Our study is conducted on a different perspective: we examine each mutation type detection independently and compare the results with the mutation score to evaluate whether the mutation score is a fair metric.

3 Technical Approach

Test Case Generation Techniques. We used Randoop [3, 14] and EvoSuite [1, 8] to automatically generate test suites. We also used manually written test suites [13] that exist on the projects that compose our experiment.

Mutation Testing Tool. We used PIT [2, 6] with its default mutation operators. These operators are designed to generate hard to kill mutants, and generating a minimal amount of equivalent mutants [4].

Case Study Applications. We selected the projects to compose our experiment based on the following requirements: 1) The project is built with Maven; 2) The project has a manually written test suite; 3) It is possible to generate test cases for the project using both Randoop and EvoSuite; 4) It is possible to generate mutants of the project using PIT.

We evaluated the above requirements for all 43 projects from the Apache Commons repository. The 10 projects that suit the requirements are described in Table 1.

Experiment Procedure. We generated 10 test suites, for each project, with each test generation tool, using their default

Table 1. Description of the projects.

Project	Version	# of Classes	LOC
BCEL	6.3	426	30812
CLI	1.4	25	2790
Codec	1.12	86	8325
CSV	1.6	14	1742
Email	1.5	21	2815
FileUpload	1.4	47	2425
Imaging	1.0	403	32607
Lang	3.8.1	238	27646
Statistics	1.0	35	2665
Validator	1.6	73	7409
Total	—	1368	119236

configuration, except for the following argument changes: Evosuite's `separateClassLoader = false` to avoid conflicts with PIT's bytecode instrumentations; Randoop's `flaky-test-behavior = DISCARD` to remove flaky tests and we also changed Randoop's `randomseed` in every execution to produce different test suites, as Randoop is deterministic by default.

From the ATSs of both tools, as well as the MTSs, we manually removed the tests that did not pass, since we wanted to analyze the test suites using mutation analysis. Having the green test suites, we ran PIT for each one of them 10 times, in order to look for nondeterminism. We found that the outputs of the executions were different just for the manual test suite of the Statistics project. In any case, we used the mean of the 10 executions to represent PIT's output.

4 Experimental Results

[RQ1] The detection ratios of each mutation type according to each test generation method are presented in Figure 1. For each mutation type, we compared the detection ratios achieved by the tests from the 3 test generation techniques using the Wilcoxon Test. We found that the MTSs are better (p-value < 0.05) than both suites generated with EvoSuite and Randoop at detecting 6 mutation types and that there is no significant evidence (p-value ≥ 0.05) of a difference between them at detecting 1 mutation type: InvertNegatives. We also found that the test suites from EvoSuite are better (p-value < 0.05) than those produced by Randoop at detecting 3 mutation types: ConditionalsBoundary, NegateConditionals, and Math; and that there is no significant evidence (p-value ≥ 0.05) of a difference between them at detecting the other 4 mutation types.

[RQ2] The mutation scores achieved for the tests from each test generation technique are presented in Figure 2. We compared the mutation scores from each technique using the Wilcoxon Test and we found that MTSs have higher mutation score than EvoSuite's and Randoop's test suites (p-value = 0.000976 in both cases). We also found that EvoSuite's test suites have higher mutation score than Randoop's test suites

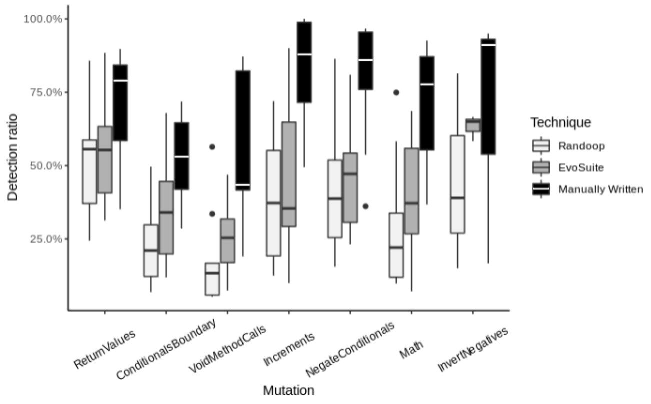

Figure 1. Detection ratios of PIT's default mutation types according to test suites from different test generation techniques.

(p-value = 0.00293). Thus, according to the mutation score, MTSs are better than the ATSs and EvoSuite's test suites are better than Randoop's test suites. Relating these results with the answer to RQ1, we found that when the mutation score of test suites from a test generation technique A is higher than the mutation score of test suites from a technique B, A's tests are better than B's tests in relation to certain mutation types and equivalent in relation to other mutation types. We also found moderate to strong positive correlation, using Spearman's Rank correlation, between the mutation types detection ratio, according to the test suites from the 3 unit test generation methods.

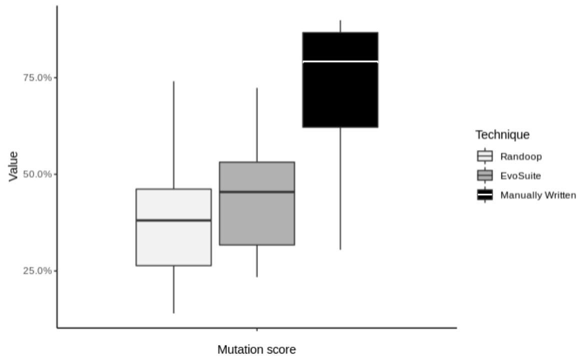

Figure 2. Mutation scores achieved for the test suites from each test generation technique.

5 Conclusions

All data used in our experiment and the statistical analysis are publicly available at https://biabs1.github.io/SRC-2019/. Our results suggest that it is fair to use the mutation score to compare test suites from different test generation techniques. The moderate to strong positive correlations between the mutations detection ratios indicate that the detection of different mutants by test suites tend to grow proportionally. However, this may also indicate duplicated mutants [7, 12].

References

[1] 2019. EvoSuite: Automatic Test Suite Generation for Java (2019). http://www.evosuite.org/evosuite/ Accessed: 2019-09-04.

[2] 2019. PITest Mutation Testing Tool for Java (2019). http://pitest.org/ Accessed: 2019-09-04.

[3] 2019. Randoop: Automatic unit test generation for Java (2019). https://randoop.github.io/randoop/ Accessed: 2019-09-04.

[4] Michael Andersson. 2017. *An Experimental Evaluation of PIT's Mutation Operators*. Bachelor thesis. Umeå University.

[5] J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is mutation an appropriate tool for testing experiments? [software testing]. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEE, Saint Louis, MO, USA, USA, 402–411. https://doi.org/10.1109/ICSE.2005.1553583

[6] Henry Coles, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. 2016. PIT: A Practical Mutation Testing Tool for Java (Demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, New York, NY, USA, 449–452. https://doi.org/10.1145/2931037.2948707

[7] Leonardo Fernandes, Márcio Ribeiro, Luiz Carvalho, Rohit Gheyi, Melina Mongiovi, André Santos, Ana Cavalcanti, Fabiano Ferrari, and José Carlos Maldonado. 2017. Avoiding Useless Mutants. In *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 2017)*. ACM, New York, NY, USA, 187–198. https://doi.org/10.1145/3136040.3136053

[8] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-oriented Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 416–419.

[9] James H. Andrews, Lionel Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. *Software Engineering, IEEE Transactions on* 32 (09 2006), 608–624. https://doi.org/10.1109/TSE.2006.83

[10] Y. Jia and M. Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (Sep. 2011), 649–678. https://doi.org/10.1109/TSE.2010.62

[11] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are Mutants a Valid Substitute for Real Faults in Software Testing?. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 654–665. https://doi.org/10.1145/2635868.2635929

[12] M. Kintis, M. Papadakis, Y. Jia, N. Malevris, Y. Le Traon, and M. Harman. 2018. Detecting Trivial Mutant Equivalences via Compiler Optimisations. *IEEE Transactions on Software Engineering* 44, 4 (April 2018), 308–333. https://doi.org/10.1109/TSE.2017.2684805

[13] J. S. Kracht, J. Z. Petrovic, and K. R. Walcott-Justice. 2014. Empirically Evaluating the Quality of Automatically Generated and Manually Written Test Suites. In *2014 14th International Conference on Quality Software*. IEEE, Dallas, TX, USA, 256–265. https://doi.org/10.1109/QSIC.2014.33

[14] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: Feedback-directed Random Testing for Java. In *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion (OOPSLA '07)*. ACM, New York, NY, USA, 815–816.