Improving Automatic Test Case Generation by predicting optimal Fitness Functions

Toon Kling
Supervisor(s): Mitchell Olsthoorn, Pouria Derakhshanfar, Annibale Panichella
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

ABSTRACT

Recently, automating test suite generation is a problem that has drown attention in both industry and academia. One of the tools used to automatically generate test suites is EvoSuite, which is a state-of-the-art tool often used in research. It uses a genetic algorithm, which seeks to maximize certain coverage criteria, such as Branch Coverage or Exception Coverage. Previous research has investigated the possibility of combining multiple coverage criteria, but there is no single combination which performs best for all classes. This paper aims to investigate if it possible to predict when coverage criteria perform best, specifically Exception Coverage, according to the characteristics of the class under test. The paper shows that there is a significant difference between the performances of these coverage criteria. The paper also shows that this difference can be predicted using a Machine Learning model with an f1-score of 0,865, when performance was measured using Mutation Score. Lastly, an exploration is made into the characteristics of the Class-under-test which cause this.

1 INTRODUCTION

Testing software is an important part of developing software, it is necessary to ensure that software functions according to the expectations of the developers. Writing test cases can take developers significant amounts of time [11]. If the time developers have to spend on testing their code can be reduced, their efficiency will be increased.

A tool that can solve this problem is EvoSuite [6]. EvoSuite is a tool that automatically generates test suites for a class-under-test (CUT) in Java. [todo why EvoSuite]It has been shown that these test cases can detect real faults in used software [1, 7]. A well-functioning test suite generation tool can help reduce time developers spend writing tests.

EvoSuite generates test suites according to various coverage criteria. These have been defined over time to optimize for certain goals, such as Branch coverage, Input coverage or Exception coverage. It has been shown that combining different coverage criteria can provide better results than using coverage criterion individually [14].

The effect of different coverage criteria has been studied by Gay [9]. They found that for different classes there are different optimal criteria or combinations of criteria, there is no single combination of criteria that performs best for all classes. This means that if it is known which (combination of) criteria performs best for a given class, the generated test suite could be better.

What has not yet been researched is if there is a way to predict which of the coverage criteria will perform best for each class. This prediction would depend on the characteristics of the class under test (CUT). If such a prediction can be made, then before test suite generation the optimal coverage criterion could be selected. This would make test suite generation more effective. For this paper there will be a focus on the coverage criteria of Branch Coverage and Branch Coverage combined with Exception Coverage.

Which coverage criteria are best for each class depends on the characteristics of the Class-under-test. The prediction of which coverage criteria will be made based on those characteristics.

The main question that will be answered is **"How effective are the different state-of-the-art fitness functions at guiding the search process towards finding bugs?"**, where the research question of this project is **"When and how does Exception Coverage increase the number of bugs detected when combined with branch coverage?"**.

To answer this question, the following sub-questions have to be answered:

- What is the improvement like when combining the BRANCH coverage criterion with the EXCEPTION criterion?
- How accurate are possible models in predicting when combining with the EXCEPTION criterion brings improvement?
- Under which class characteristics does combining with the EXCEPTION coverage criterion give improvements?

The contribution of this paper is that it will investigate which characteristics of the CUT are relevant for this research, and whether this connection exists.

The contribution of this paper is that it shows that there is a statistically significant difference between the performance of the BRANCH coverage criteria and it combined with EXCEPTION. Then it is shown that it is possible to make a prediction based on the characteristics of the Class-under-test. Lastly there is an exploration of which characteristics of the Class-under-test contribute to the combination of BRANCH and EXCEPTION coverage criteria performing well.

First the paper will give some background information. Next the methodology that is used will be discussed. After that, there will be a section describing the results. Then the threats to validity will be discussed. There will also be a section on responsible research. The paper will finish with the conclusions and future work.

1.1 Background

This section will give some background information. First there will be some more detail given on EvoSuite, coverage criteria and fitness functions. Then an overview of different possible coverage criteria will be given. Lastly there will be an explanation of some of the class metrics that will be used.

1.1.1 EvoSuite. As mentioned in the introduction, EvoSuite is a tool for generating test suites. It was chosen because it is a state-of-the-art test suite generation tool, that is often used in research.

EvoSuite generates test suites that optimize a certain *coverage criterion*, or a combination of multiple *coverage criteria*. These coverage criterion are for example Branch Coverage (percentage of branches covered by a test) or Exception Coverage (amount of possible exceptions covered by a test). The coverage criteria show the quality of the current test suite. These coverage criteria form a *meta-heuristic* to guide the search of EvoSuite.

Covering the entire coverage criterion involves generating a test suite that covers all the parts of the criterion. In the example of Branch Coverage as the coverage criterion, each branch must at least have a test case covering it. The distance between a certain test

case and it's target branch, is given by a fitness function. The fitness function of a test depends on the coverage criterion or combination of criteria chosen. The goal of the tool is then to optimize this set of fitness functions.

Optimizing these functions in EvoSuite is done by a genetic algorithm, DynaMOSA, which was proposed by Panichella *et al* [12]. DynaMOSA is a many-objective solver that optimizes the different fitness functions. Since there is a prohibitively large amount of fitness functions to optimize, DynaMOSA utilizes the hierarchy between the coverage criteria to select a subset of coverage targets to focus on. Since recently DynaMOSA has also been able to efficiently search through a combination of coverage criteria [13].

1.1.2 Coverage Criteria. EvoSuite comes packed with a standard list of coverage criteria. In this section the coverage criteria that are relevant for this paper will be covered [8]. Some of these coverage criteria are used in the papers of other peers:

(1) **BRANCH**. Branch coverage is attained when all branches, possible paths in relation to control flow, are executed.
(2) **CBRANCH**. Branch coverage is attained when all branches, possible paths in relation to control flow, are executed.
(3) **EXCEPTION**. Exception coverage means that the more exceptions that are thrown in the class under test (CUT), the better.
(4) **WEAKMUTATION**. The amount of mutants that are detected. Since weak mutations is used, these mutations do not need to reach program output.
(5) **OUTPUT**. This coverage criteria rewards a higher diversity in the output. Distance can be used to define the distance between the different output types, for example returning all possible subclasses or distance in numeric terms.
(6) **INPUT**. Similar to output coverage.

1.1.3 Class metrics. The goal is to find which characteristics of the CUTs influence the effectiveness of different coverage criteria. An often used set of metrics that give information about a class are the metrics developed by Chidamber & Kemerer [4]. There also exists an extension of these metrics, developed by Aniche [3], which computes a total of 49 class metrics.

1.1.4 Model. This section will give background information on the model that will eventually be used. It will only give background information that the average reader is not expected to know. Since the choice of model is not yet made, this part remains empty.

1.2 Related work

As mentioned, Gay [9] has investigated the results of combining fitness functions. They have found that there are different optimal fitness functions for each class, however they have made no attempts to predict this.

Coverage criterion selection has been done by Almulla *et al.*[2], who propose an adaptive algorithm where fitness function selection is another step of optimization. The results are promising, but a reinforced learning model will always contain some overhead. The method proposed in this paper would not have such a significant overhead.

2 METHODOLOGY

In this section there will be an overview of the methodology that was used to answer the research question. First there will be a description of the dataset that was used. Then the method for answering the first research question will be covered. Secondly, the metrics that were collected from the classes are coverd. Then it is discussed how the data was cleaned and balanced. Next the paper describes how the second research question was answered by developing a model. Then it answers how to measure the performance of that model before ending with the last research question.

2.1 Data collection

The set of classes that was used was the SF110 corpus of classes [7], over which the test suites and their performance evaluation were generated. The SF110 corpus is a collection of 100 random classes combined with the 10 most popular classes from SourceForge. This set of classes were chosen because they are a large representative sample of open-source Java classes.

On this dataset test suites were generated using EvoSuite for each class. These classes were scored by the Branch Coverage which EvoSuite computed and the Mutation Score which was computed using PITest[5]. EvoSuite was run for each class, for each criterion and for each time budget. The time budgets were 60 seconds, 180 second and 300 seconds. EvoSuite was run 10 times for each class and each criterion. The median value for these 10 runs was taken as the Branch Coverage value for that class and criterion. Note that BRANCH refers to using the BRANCH coverage criterion only, and BRANCH;EXCEPTION refers to using BRANCH and EXCEPTION coverage criteria combined. Only the criteria BRANCH and BRANCH;EXCEPTION were investigated for this paper. All criteria for which EvoSuite was run were:

- BRANCH
- BRANCH;CBRANCH
- BRANCH;EXCEPTION
- BRANCH;INPUT
- BRANCH;OUTPUT
- BRANCH;WEAKMUTATION
- default (combines all coverage criteria)

A sample of the data this produced can be seen in Table 1. Here the TARGET_CLASS is the class that the data was taken from, the round refers to the 10 times that EvoSuite was run and the criterion refers to which criterion this run was for. TARGET_CLASS has been shorted for ease of reading. The table also contains the Branch Coverage that this resulted in.

Besides only the Branch Coverage for each test suite, the Mutation Scores were also computed by using the PITest tool. The Mutation scores were generated for each class and each criterion, but only for the 60 seconds time budget.

2.2 RQ 1 - What is the improvement like when combining the BRANCH coverage criterion with the EXCEPTION criterion?

To answer the first research question, the median results in Branch Coverage and Mutation Score was computed for each class and

project.id round	TARGET_CLASS criterion	BranchCoverage
24_saxpath 3	[...].XPathLexer BRANCH;CBRANCH	0.878099
jdom 3	[...].XMLOutputter BRANCH;EXCEPTION	0.951613
scribe 10	[...].Verifier BRANCH;OUTPUT	1.000000
twitter4j 8	[...].TwitterException default	0.829268
59_mygrid 10	[...]._AvailableJobsResponse BRANCH;EXCEPTION	0.821429

Table 1: A random sample of EvoSuite data, $n = 5$.

criterion, so taking the median of the 10 rounds. The data resulting frmo this has been used to compute the number of classes where each criterion performs best, and graphs displaying the average performance of each criterion. This has been done for both Branch Coverage and Mutation Score. Based on this, it is possible to see which criterion or combination of criteria performs best.

2.3 Metrics

This paper uses the *CK* tool mentioned in the background, developed by Aniche [3], to compute the metrics for each class. These metrics have been used as features of each class in order for the model to use for predictions. To this end, the metrics have been normalized.

2.4 Data cleaning

With the dataset that was computed, some metrics contained Not-A-Number (*NaN*) values. Since these values are undefined, it is best to define a way to replace them, or not to use the classes where they occur. There were three code metrics where *NaN* values occurred.

2.4.1 LCOM.* This was the first code metric which contained *NaN* values. These *NaN* values were caused because the method that computes LCOM* divides by the number of methods as one the steps to compute it. Whenever a class does not have any methods, this results in a division by zero. The tool [3] was written in Java, which means that the division by zero eventually resulted in the *NaN* values. A class with no methods should have a high cohesion, so these *NaN* values were replaced with 1.

2.4.2 TCC & LCC. These metrics compute Tight and Loose class cohesion. This is computing by dividing the amount of connections of a class by the total amount of possible connections of that class, the amount of possible connections is computed by $N * (N - 1)$, where N is the number of visible methods. Whenever $N < 1$, TCC and LCC are undefined because a division by zero would occur so the metric returns -1. However, whenever $N = 1$, a division by zero also occurs, which leads to the *NaN* values being present in the data. Therefore, for these metrics *NaN* was replaced with -1, which is the value they should have.

2.5 Imbalanced data

Learning from unbalanced data can reduce the performance of predictive models [10]. Unbalanced data occurs when the occurrence of different categories in the data is unequal, some data appears less often than average, while some categories appear more often than average.

Imbalanced data is often resolved by either artificially adding more datapoints to the rarer category, which is called *oversampling*, or by removing datapoints from the more common category, which is called *undersampling*. Since *undersampling* reduced the amount of available data, this paper will randomly oversample the training data.

2.6 RQ 2 - How accurate are possible models in predicting when combining with the EXCEPTION criterion brings improvement?

To answer this research question, a model was developed. This model should try to predict which criterion, BRANCH;EXCEPTION or BRANCH, performs better for a given class. A separate label was introduced for if both criteria produce the same results. Therefore the labels that should be predicted are *better*, *equal* or *worse*, representing performance of BRANCH;EXCEPTION compared to BRANCH.

The features on which the prediction should be based are the features that are mentioned in the metrics section.

For predicting when a higher Branch Coverage or Mutation Coverage was achieved, a model was developed. For both of these questions a Random Forest Classifier was used. This classifiers trains a number of decision trees on samples of the training data. The final classification is an average of the classification of the individual decision trees.

2.7 Measuring performance

While the results of the classification use 3 labels, *better*, *equal* and *worse*, the prediction should give a boolean answer: which criterion performs better. A choice should be made which criterion to use in the case that the model predicts an equal outcome. Since the model will not predict perfectly, this choice will have an effect. It was chosen that an *equal* prediction would be interpreted as using the BRANCH;EXCEPTION criterion. This reduces the amount of final labels from 3 to 2, which makes measuring the performance of the model simpler.

Since the dataset is imbalanced, as discussed in section 2.5, a simple accuracy measurement would not be helpful. Instead of an accuracy score, often picked metrics to evaluate imbalanced learning problems are *recall* and *precision* or a score which combines both, *F-Measure* [10]. Here *precision* refers to how often a prediction for the *true* class was actually correct, and *recall* gives a measure of how often an actually correct class was labeled as *true*. F-Measure combines these scores according to the following formula:

$$\text{F-Measure} = \frac{(1+\beta)^2 \cdot Recall \cdot Precision}{\beta^2 \cdot Recall \cdot Precision}$$

F-Measure with $\beta = 1$ was used as one way to measure the performance of the models. This is referred to as the *f1-score*.

Another way of measuring performance is to simulate what the results of running EvoSuite with criterion selection from this model would be. As mentioned, the dataset contains the Branch Coverage and Mutation Score for each class for each criterion. This allows the computation of the average Branch Coverage and Mutation Score for each criterion. But it also allows the computation of this average, when the selection is made which criterion to use for each class based on the prediction of the model. For comparison, this measurement will also contain the average result with a fake model that has perfect predictions. Therefore, this measure contains 4 datapoints, each representing the average Branch Coverage or Mutation score, for the following criteria: BRANCH, BRANCH;EXCEPTION, model-prediction, perfect-prediction.

The advantage of this way of measuring the performance of the model is that it corresponds to a real-world application of EvoSuite with this model. Using this model tells us whether it is possible to improve EvoSuite using a prediction like this. In order to provide an accurate measurement the the data was split using K-Fold cross-validation, with 5 splits.

Whether the model with selection outperforms the criterion is relevant information. In the case that this was very close and not conclusive, the difference between the selection with the model and the usage of that criterion was computed over 10 runs of training the model.

2.8 RQ 3 - Under which class characteristics does combining with the EXCEPTION coverage criterion give improvements?

This research question concerns the characteristics of the classes that contribute to which criterion performs best. This research question will be answered by exploring which characteristics of the class correspond to the BRANCH;EXCEPTION combination performing well, as well as exploring the Decision Trees comprising the Random Forest classifier that was developed for research question 2.

3 RESULTS & ANALYSIS

This section will give an overview of the results. The results will be given per Research Question.

3.1 RQ 1 - What is the improvement like when combining the BRANCH coverage criterion with the EXCEPTION criterion?

As can be seen in table 2, there is a significant number of classes where BRANCH;EXCEPTION outperforms BRANCH, (44% of the time). There are cases where this is not the case or where the results are the same(56%). As can be seen in figure 1 and 2, there is a significant average improvement when using BRANCH criterion as opposed to using the BRANCH;EXCEPTION criterion. However, it should be noted that there is also a significant amount of classes where the BRANCH criterion performs best. This means that for both Branch Coverage and Mutation Score a greatest performance can be achieved if it is correctly predicted which criterion performs best.

Metric	Total	*B* better	*B;E* better	equal
BranchCoverage	338	94	79	165
MutationScore	321	71	140	109

Table 2: For how many classes the different criterion outperform eachother. Comparing *BRANCH* (B) to *BRANCH;EXCEPTION* (B;E for both metrics that were collected.

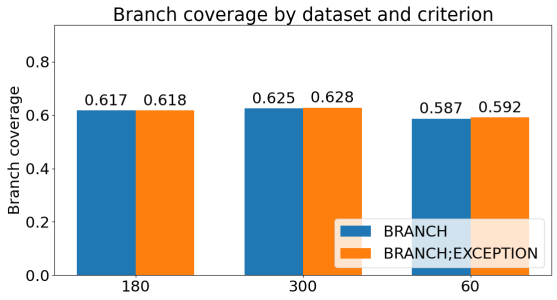

Figure 1: Average Branch coverage for each dataset and each criterion.

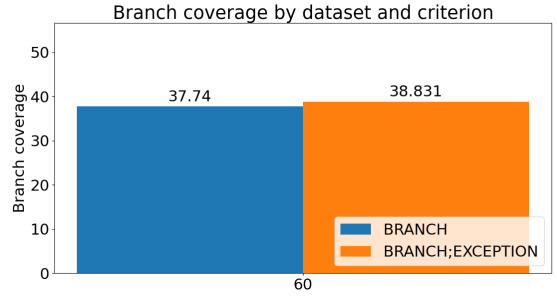

Figure 2: Average Mutation Score for both criteria.

3.2 RQ 2 - How accurate are possible models in predicting when combining with the EXCEPTION criterion brings improvement?

In Figure 3 there is a comparison of the results of different criteria. As mentioned, the figure shows the resulting Branch Coverage had EvoSuite been run on the test classes according to the various criteria. The model achieved an *f1-score* of 0,793.

In the figure it can be seen that using the predictions from the model outperforms a naive approach of picking either of the criteria themselves. However, this experiment was run 10 times, and the the model outperforming BRANCH;EXCEPTION was not consistent, of the total of 10 runs, only 5 times the model predictions outperformed selecting BRANCH;EXCEPTION as a criterion. This means that the results are not consistent.

The experiment was also run for Mutation Score prediction, as can be seen in Figure 6. Here, oversampling did not have a positive

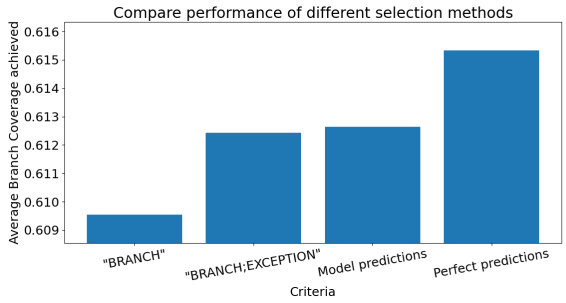

Figure 3: Average Branch Coverage comparison for each possible criterion.

effect on the outcome so it was not done. As can be seen, the Model predictions outperform both criteria. This experiment was repeated 10 times as well, with the model outperforming both BRANCH and BRANCH;EXCEPTION individually 8 out of 10 times. The model seen in the figure achieved an f1-score of 0,865.

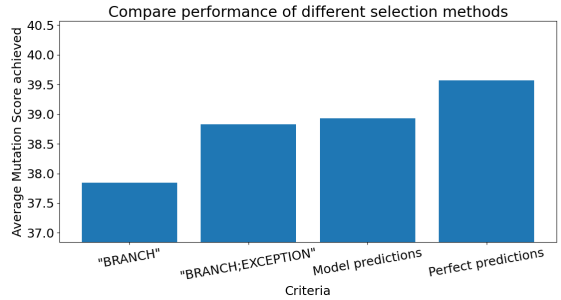

Figure 4: Average Mutation Score comparison for each possible criterion.

3.3 RQ 3 - Under which class characteristics does combining with the EXCEPTION coverage criterion give improvements?

Firstly in table 3 and 4 are the 10 class metrics that correlated the most with whether BRANCH;EXCEPTION outperforms BRANCH for both Branch Coverage and Mutation Score. These results show that there are definitely some characteristics which are correlated with the performance, however individually they do not correlate significantly.

For figure 4, which contains the metrics and their correlation to Branch Coverage, it can be seen that there is often a significant, negative correlation. The metrics which have this negative correlation can be seen as metrics indicative of a more complex class.

Class metric	Correlation
synchronizedMethodsQty	-0.107576
mathOperationsQty	0.097993
fanout	0.096586
cbo	0.096586
maxNestedBlocksQty	0.090270
assignmentsQty	0.088040
variablesQty	0.086791
rfc	0.086417
anonymousClassesQty	0.082053
stringLiteralsQty	0.081770

Table 3: Most correlating metrics for Mutation Score

Class metric	Correlation
uniqueWordsQty	-0.276232
comparisonsQty	-0.263137
rfc	-0.226523
assignmentsQty	-0.217081
visibleMethodsQty	-0.216861
variablesQty	-0.212898
publicMethodsQty	-0.192095
staticFieldsQty	-0.187837
totalMethodsQty	-0.187225
nosi	-0.184247

Table 4: Most correlating metrics for Branch Coverage

4 VALIDITY

4.1 Threats from Class Selection

The dataset that was used in this paper is the SF110 corpus of classes. If there is a bias in the selection of these classes this could influence whether the results could be generalized. However, the SF110 corpus of classes contains a random selection of 100 open-source projects from Sourceforge, which means that there was no bias in their selection. It does contain the 10 most popular popular projects on Sourceforge as well, however this is not a significant enough number to threaten the results provided. There is a bias present towards open-source projects written in the Java language. Whether these results also apply to other languages is not investigated.

4.2 Threats from EvoSuite

The test suite generation from EvoSuite is by it's nature a stochastic process. The results for each generation could differ signficantly from the others. The threats to validity from this fact were alleviated by running the algorithm 10 times for each class, and taking the median of those values as the result. This reduces the risk of random chance interfering with the results.

4.3 Threats from Mutation Score

The computation of mutation scores for each generated test suite is by it's nature also a stochastic process. Here too, 10 runs were

done for each class. The result that was worked with is the median of these 10 runs.

4.4 Threats from Random Models evaluation

The models that were developed during this paper are made up of stochastic processes, their performance varied significantly. Performance metrics were made less variable by using K-Fold cross validation with 5 folds. During answering of the second research question, where it was relevant whether the model-selected coverage criterion outperformed "BRANCH;EXCEPTION", the experiment was repeated 10 times to get a conclusive answer.

5 ETHICS

This paper concerns a tool which is used by developers to write tests. It cannot influence humans or the environment directly. Care should always be taken when relying on automatically generated test cases.

The dataset of this paper, SF110, is a collection of open-source projects. EvoSuite, PITest and CK, which computes the class metrics, are all open-source tools. This allows the analysis that was done in this paper.

The results in this paper can be reproduced by running the code. The code will be released and be made accessible.

6 CONCLUSIONS AND FUTURE WORK

Criteria for guiding automated test suite generation algorithms like EvoSuite have been studied recently. Two of these criteria, or combinations of criteria, are BRANCH and BRANCH;EXCEPTION, which this paper focuses on. In this paper, it is shown that it is possible to predict whether BRANCH or the combination performs better.

To this end, first it is shown that there is a significant difference in performance between these characteristics. This is based on an evaluation of the SF110 Corpus of classes, of which test suites have been generated using EvoSuite, and class metrics using the *CK* tool, developed by Aniche [3].

Based on this data, a Random Forest model has been developed to predict which criterion performs best, where performance is measured by Branch Coverage or Mutation Score. It has been shown that this model could achieve a higher Mutation Score compared to each of the criteria individually, with an *f1-score* of 0,865. The results for achieving a higher Branch Coverage were promising but not conclusive.

In order to find which characteristics contribute to which criterion performs best, a list of correlated metrics has been produced. For Branch Coverage, the most significant correlations were negative. In other words, Exception Coverage achieves higher results with low metrics. Generalizing all metrics it can be said that a class which scores low on all of them is often "simpler". Therefore Exception Coverage will only really achieve improved Branch Coverage on simpler classes. Since Exception Coverage does not necessarily automatically lead to better Branch Coverage, this does not necessarily have consequences for real fault-detection capability.

For mutation score there are more metrics which have a significant positive correlation with a good result for BRANCH;EXCEPTION criterion as opposed to BRANCH criterion. The list of metrics can be of interest to see which characteristics contribute to the BRANCH;EXCEPTION performing well. A developer could use this information to make a more informed decision on which coverage criteria to use.

For future work, this paper only compares two coverage criteria or combinations of, while a real implementation with EvoSuite would require to make this prediction for more combinations. This paper has not been able to predict when BRANCH;EXCEPTION achieves a better Branch Coverage, only when it achieves a better Mutation Score. More analysis into which characteristics contribute of classes contribute to the difference in performance could also be performed.

REFERENCES

[1] M Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Janis Benefelds. 2017. An industrial evaluation of unit test generation: Finding real faults in a financial application. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 263–272.

[2] Hussein Almulla and Gregory Gay. 2022. Learning how to search: Generating effective test cases through adaptive fitness function selection. *Empirical Software Engineering* 27, 2 (2022), 1–62.

[3] Maurício Aniche. 2015. *Java code metrics calculator (CK)*. Available in https://github.com/mauricioaniche/ck/.

[4] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[5] Henry Coles, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. 2016. Pit: a practical mutation testing tool for java. In *Proceedings of the 25th international symposium on software testing and analysis*. 449–452.

[6] 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*. 416–419.

[7] Gordon Fraser and Andrea Arcuri. 2014. A large-scale evaluation of automated unit test generation using evosuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 1–42.

[8] Gregory Gay. 2017. The fitness function for the job: Search-based generation of test suites that detect real faults. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 345–355.

[9] Gregory Gay. 2017. Generating effective test suites by combining coverage criteria. In *International Symposium on Search Based Software Engineering*. Springer, 65–82.

[10] Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.

[11] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons.

[12] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.

[13] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Incremental control dependency frontier exploration for many-criteria test case generation. In *International Symposium on Search Based Software Engineering*. Springer, 309–324.

[14] José Miguel Rojas, José Campos, Mattia Vivanti, Gordon Fraser, and Andrea Arcuri. 2015. Combining multiple coverage criteria in search-based unit test generation. In *International Symposium on Search Based Software Engineering*. Springer, 93–108.