Test Suite Reduction Using Machine Learning

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

ABSTRACT

Creating unit tests manually is labor-intensive and costly. Against this background, a bunch of automated tools for generating unit tests have emerged. Random testing tools are easy to implement, but always generate a large number of redundant test cases. For these reasons, researches on Test Suite Reduction (TSR) come into being, and most existing TSR methods use heuristic algorithms. However, these methods may consume much time and the optimization results are not satisfying. Therefore, recent studies tend to use Machine Learning (ML) techniques to help reduce the test suite size. In our experiment, we study Randoop, which is an automated test generation tool using random testing, and propose two test case filtering models based on ML technology: (i) Model for coverage prediction of test cases; (ii) Model for similarity calculation between different test cases. Furthermore, we integrate these two ML models with Randoop. If the code coverage of a newly created method sequence is lower than a fixed threshold, we remove it to reduce test suite size. Additionally, we combine the model for similarity calculation with k-means clustering to select the most representive cases from different clusters, thus reducing redundancy. In summary, our approaches can reduce test suite size by 20% on average, improve readability under the condition of not substantially deviating from coverage or mutation score obtained from the initial test suite. In this paper, we gain deeper insight into TSR approaches based on ML to improve the quality of test suites generated by random testing tools.

CCS CONCEPTS

• **Software Testing → Automated Unit Test Generation Tools**;
• **Random Testing**; • **Machine Learning**;

KEYWORDS

Random Testing, Machine learning, Test Suite Reduction

1 INTRODUCTION

Software testing is an essential task in the software development process. Unit testing has an important impact on the effectiveness of software testing and has proven its value in detecting real faults. However, writing high-quality unit tests is time-consuming and laborious. The emergence of automated unit test generation tools alleviates this problem, but there's still the issue of generating low-quality test cases. We research three different test generation techniques in our paper:

- **Search-based Software Testing** [3] uses search-based optimization algorithms to automatically search for test data, so as to maximize code coverage and minimize cost. EvoSuite [11] and JTeXpert [22] are tools that use search-based approach.
- **Symbolic Execution** [6] analyzes the code with the help of formal semantics of the program. This technique can be used to automatically generate unit test cases. KLEE [5] is a currently mature symbolic execution tool.
- **Random Testing** [3] selects method calls and parameters randomly to incrementally generate method sequences. Randoop [19] uses random testing technology to automatically produce unit test suites.

For a specific class under test (CUT), the test suite generated by random testing tools always contains a great deal of test cases, which far exceeds the number of cases generated by automated tools using other technologies. However, too many test cases affect the overall readability. Additionally, the larger the test suite, the higher the test budget and time requirements, which is not conducive to the efficiency of software testing. **Test suite reduction (TSR) is considered to be a potentially effective approach to deal with the problem of test suite size.** TSR can eliminate the redundant method sequences and diminish the cost of executing, managing and analyzing test cases.

To address this problem, many researchers apply heuristic methods to find a subset of test cases for reducing test suite size and decreasing regression testing cost in software continuous evolution. Most existing approaches analyze the similarity between the current test case and the previous test cases to determine whether the current one is redundant. The coverage-based reduction approach [15] executes test cases, and checks coverage of each test case to seek out redundancy without decreasing the overall code coverage, but this technique consumes too much time and cost. Machine learning algorithms can be used for regression prediction and similarity calculation, which has a certain tolerance to wrong data. ML has higher efficiency than traditional methods that calculate coverage directly, and it can be well applied to code similarity calculation.

We compare the test suite size generated by two search-based testing tools and a random testing tool Randoop, and find that the test suite size generated by Randoop is much larger than the others. Randoop is a state-of-the-art tool based on feedback-directed mechanism, which eliminates some useless test inputs by restricting access to objects. However, this mechanism cannot achieve the expected goal of TSR, Randoop still generates a huge number of redundant test cases on a given class under test (CUT). To help optimize the test suite reduction process of randomized test generation tools like Randoop:

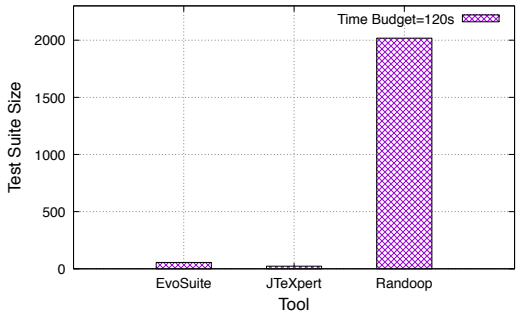

Figure 1: Test suite size of EvoSuite, JTeXpert and Randoop when time budget is 120 seconds

1) We propose a method using ML technology to predict code coverage of test cases generated by Randoop in real time, and then remove cases with very low predicted coverage.

2) We propose a ML approach that measure the similarity between test cases. Additionally, the k-means clustering approach [17] is proposed for test suite reduction.

2 RELATED WORK

Many researchers have noticed that the number of test cases generated by random testing tools is too large. Chitirala et al. [8] analyze and compare the test suite size generated by EvoSuite [11] and Tpalus [30]. Tpalus is a random test generation tool based on Randoop with some extensions. Experimental results show that test suite generated by Tpalus is much larger than EvoSuite, while EvoSuite performs test suite minimization very well. They also point out that large test suite is hard to manage and understand. To help address this problem the software testing research community has responded with a wealth of research that aims to contribute for test suite reduction. Harrold et al. [14] apply the representative set algorithm to the selection of a subset of a given test suite. This approach can reduce the number of test cases on the basis of ensuring coverage. Jayagarl et al. [15] present a tool called GenRed, which uses a coverage-based reduction technique with measuring code coverage and a sequence-based reduction technique without executing test cases when it comes to TSR. Khan et al. [16] conduct a detailed investigation and comparison of randomized unit testing based TSR tools and emphasize their strengths and shortcomings. However, their research illustrates that the existing TSR methods are lacking in the effectiveness of eliminating redundancy, while the cost is very high.

There are many examples of optimizing test suites generated by automated test generation tools combined with machine learning algorithms. Taylor et al. [26] propose a multi-objective search-based technique based on machine learning by inferring finite state machines (FSMs) from execution traces for test suite reduction. Briand et al. [4] present an automated methodology, based on decision tree, to analyze the weaknesses of test suites so as to be able to iteratively refine and reduce them. Nour et al. [7] introduce a machine learning based algorithm for test suite reduction that combines k-means clustering with binary search. Daka et al. [9] present a ML model to quantify and evaluate the readability of test cases generated by EvoSuite. Results show that their work can make test cases more readable without decreasing code coverage. Giovanni et al. [13] propose ML approaches to predict the branch coverage for a given CUT achieved by automated test generation tools (EvoSuite and Randoop).

The use of machine learning for similarity measurement has been proposed recently. Gomaa et al. [12] discuss three approaches on text similarity: string-based, corpus-based and knowledge-based similarity measures. Ye et al. [29] propose a novel neural-based code similarity system to identify similar codes in the program. Zhao et al. [31] present a deep learning model that measures code functional and syntactics similarity. Their researches show that using machine learning technology to calculate the similarity of test cases is a meaningful option.

3 PROBLEM FORMULATION

We use the benchmark selected in Search Based Software Testing (SBST) 2017 [21] as CUTs which include 70 classes from eight libraries in our experiment, and compare the average test suite size generated by two search-based tools EvoSuite and JTeXpert, and a random testing tool Randoop when time budget is 120 seconds. As shown in Figure 1, the results show that the number of test cases generated by Randoop is much higher than that of EvoSuite and JTeXpert while the coverage or mutation score of Randoop is worse than the others. Randoop can generate approximately two thousand test cases for a CUT on average, however, the code coverage of test suites generated by Randoop is not high though containing too many cases, and most of them do not help improve coverage. The experimental results also show that the code coverage of many test cases generated by Randoop is zero. Additionally, the content of some test cases is very similar. This indicates that there exists a lot of redundant test cases. Excessive cases affect the readability, and it takes a large amount of time to execute test cases.

Random test generation tools select inputs at random from a program's input space, and automatically generate unit test cases in JUnit format for a CUT. We take a state-of-the-art feedback-directed random testing tool Randoop as an objective for research, Randoop generates a great quantity of test cases, including many redundant cases. Existing TSR techniques spend much time to measure code coverage and similarity. Machine learning has a wide range of applications in data prediction and similarity calculation, we can take advantage of its characteristics to the process of test suite reduction in a more effective way. ML algorithms can achieve accurate prediction results, and are more efficient than program analysis when measuring code coverage. Some test cases generated by Randoop contribute very little to code coverage, which can be directly removed. In summary, we attempt to present two machine learning models, one for predicting code coverage of test cases, and one for identifying the similarity between test cases. We hope that the test suite size can be reduced by introducing these two models to the test generation process of Randoop.

4 METHODOLOGY

In our paper, we train two ML models to remove redundancy by predicting code coverage and measuring test case code similarity.

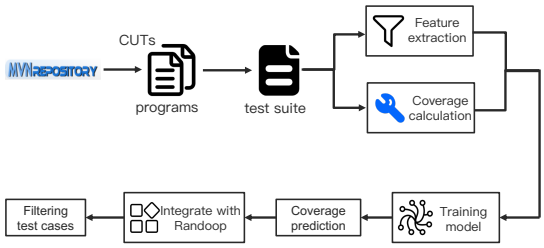

Figure 2: Possible development flow of model for predicting code coverage

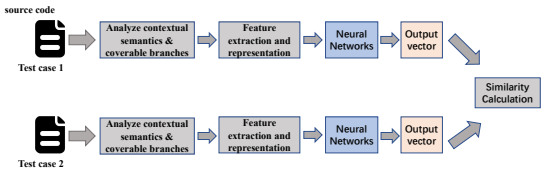

Figure 3: Possible workflow of model for similarity judgement

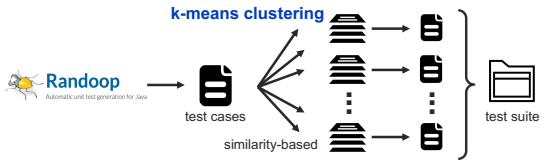

Figure 4: The possible process of similarity-based clustering algorithm for test suite reduction.

4.1 Model for Coverage Prediction

Figure 2 shows the workflow of training model for code coverage prediction. First, we extract features to represent the complexity of test case code. Second, we apply different ML algorithms to train models for predicting code coverage. Then, we evaluate their performances in terms of Mean Absolute Error (MAE) and average accuracy. We choose a ML model with the best performance to integrate with Randoop in our future work, and this model can be used to predict code coverage of test cases generated by Randoop. Finally, we remove test cases with very low predicted code coverage (less than 0.0001) in real time, thereby reducing redundancy, curtailing the number of cases in a test suite and improving readability.

The CUTs selected for our experiment come from 6 different open source projects: Fastjson, JavaMail, Google Dagger, ZXing Core, JSoup Java HTML Parser, Google Guava. We filter test cases and remove many illegal items. For example, the code coverage of an entire test suite for a specified CUT is zero. This may be caused by non-public or abstract CUTs. These data may affect the correctness of dataset. Finally, we obtain a dataset containing approximately 10,000 test cases.

We make use of abstract syntax tree (AST) and regular expression method to analyze the source code of test cases. The selected features include lines of code, Halstead complexity measures [10], number of keywords, assertions, exceptions and some indicators related to the CUTs and methods under test. Finally, we extract 24 features by analyzing the characteristic of test case.

In order to conduct a comprehensive investigation and obtain the best model, we select 5 different algorithms from the Python's scikit-learn Library for comparison using 10-cross fold validation approach. We compare five regression ML algorithms, and use grid search to adjust the parameters. These algorithms can be briefly described as following:

- **Support Vector Regression**(SVR) [23] optimizes the model by maximizing the width of the interval band and minimizing the total loss.
- **Theil-Sen Regression** (TSR) [27] is a generalized robust regression linear model, and it has strong robustness to outliers.
- **Huber Regression** (HR) [25] is also a generalized robust regression linear model, and is tolerant to data containing outliers.
- **Decision Tree Regression** (DTR) [24] can be used to solve regression problems. It is an algorithm that analyzes and calculates historical data to predict new data.
- **Multi-layer Perceptron** (MLP) [20] is an implementation of neural network in scikit-learn library, which can realize regression predictions.

Test cases that do not contribute to code coverage should be removed. Applying ML model for prediction can save much time compared to directly calculating code coverage, and achieve a high prediction accuracy.

4.2 Model for Similarity Calculation

We refer to the MISIM System [29], and design a machine learning model to measure the similarity of test cases. Figure 3 shows the possible workflow of model for similarity judgement. We hope to design a component that can perceive contextual semantic information of test cases and analyze branches that test cases may be covered. And we vectorize this information and use it as the input of our neural network. When it comes to neural networks for similarity scoring, we compare the following three network structures: (i) Graph Neural Networks (GNN) [28], (ii) Convolutional Neural Networks (CNN) [1], (iii) Recurrent Neural Networks (RNN) [18]. We train a model which can generate corresponding output vectors for the given inputs. Additionally, we use a similarity measurement method to calculate the score of similarity. Among several test cases that are too similar, we only need to save one of them.

Clustering algorithms are applied in many research areas such as image processing, data analyses, or pattern recognition. In the context of this paper, we adopt the k-means clustering algorithm for data reduction, more specifically for test suite reduction. First, k initial cluster centroids are chosen, and test cases with the highest similarity of each centroid are iteratively assigned in the centroid's cluster. Then, we choose one representive test case from each cluster to obtain a reduced test suite. Figure 4 shows the possible process of this strategy.

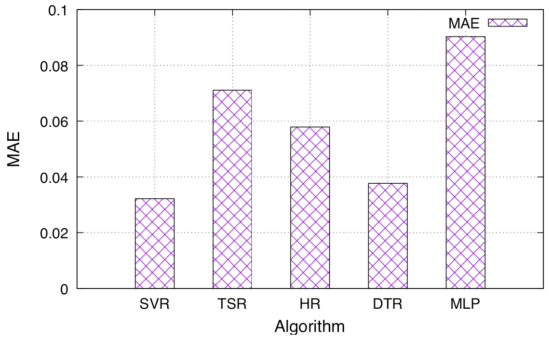

Figure 5: The MAE comparison of 5 algorithms, the smaller the value, the better the fitting result.

5 EXPECTED OUTCOMES

Our experiment aims at answering the following research questions:

- **RQ1**: Which algorithm performs best? How high is the accuracy and MAE of the model for coverage prediction?
- **RQ2**: Can the model for similarity calculation give a more accurate measurement than other current methods?
- **RQ3**: Can the test suite size generated by Randoop integrated with two models be reduced while keeping the code coverage unchanged?

RQ1: The SVR algorithm performs best among the five concluded ones. The SVR model achieves the lowest MAE. It can be seen from Figure 5 that the SVR model has the smallest loss value and the best prediction effect. The accuracy rate is expected to reach more than 95%. Therefore, we choose the SVR model to integrate with Randoop.

RQ2: Model for similarity calculation is able to give a more accurate measurement than other methods. We first compare several different network topologies. The experimental results will show that the outcome of using GNN is the best, and its area under precision-recall-gain curve can reach more than 99% with the mean average accuracy greater than 90%. We compare our system with other code similarity systems: DeepSim [31] and code2vec [2]. As a result, our model is expected to have the highest accuracy and can effectively calculate the similarity between test cases.

RQ3: After integrating models into the test generation process of Randoop, the test suite size is reduced while the overall code coverage is almost the same as before. We integrate these two models into the test reduction process of Randoop. The experimental results show that Randoop integrating with two machine learning models can achieve TSR efficiently, and the test suite size is expected to be reduced by 20% on average compared to the initial test suite.

6 DISCUSSION

Construct Validity. There may be some noises in removing test cases whose coverage is lower than a certain threshold. Some test cases have very low coverage, but they may cover new branches or new lines. However, it is worth noting that Randoop generates method sequences incrementally, branches covered by short test cases are mainly covered by other long test cases, so we can remove them directly without decreasing the overall code coverage.

External Validity. We train our model for coverage prediction with a dataset of 6 different open source projects. Nevertheless, a larger dataset can increase the applicability and generalizability of our experiment results. Additionally, we only compare RNN, CNN and GNN in our paper for the selection of for similarity calculation model, so we can choose more neural network topologies to find the most accurate similarity calculation model.

Future Work. Our models are only applied on Randoop in this paper, and in the future, we can try them on other randomized test generation tools.

7 CONCLUSION

Software testing is an essential part of software engineering. However, writing test cases manually requires a lot of manpower. The emergence of automated test generation tools alleviates this problem, but there is still room for optimization. Machine learning techniques can be applied to such tools to improve the quality of automatically generated unit test suites. Randomized test generation tools generate too many redundant test cases. Most existing test suite reduction approaches use heuristic algorithms to determine the similarity and inclusion relationship between test cases, thus identifying whether the current test case is redundant. However, these approaches require too much cost to calculate the code coverage in order to ensure it remaining unchanged.

In this paper, we take Randoop, which uses random testing, as our research object. Moreover, we propose two machine learning models integrated with Randoop, one for coverage prediction, and one for similarity calculation. We use the model for coverage calculation to predict coverage, and remove cases with code coverage under a fixed threshold for the aim of reduce redundancy. Additionally, we use a ML model for similarity calculation combined with k-means clustering algorithm to reduce test suite size. Futhermore, our proposed approach can achieve the goal of test suite reduction without decreasing the overall coverage or only small fluctuations when integrated with Randoop in future experiment. The results illustrate that using machine learning models can be an effective and promising way for test suite reduction and eliminating redundancy.

ACKNOWLEDGMENTS

This work was supported in part by National Key R&D Program (Grant No. 2017YFB1001801), the National Natural Science Foundation of China (Grant Nos. 61690204) and the Collaborative Innovation Center of Novel SoftWare Technology and Industrialization, Jiangsu, China.

REFERENCES

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. 2017. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 1–6.
[2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.
[3] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil Mcminn, Antonia Bertolino, et al. 2013. An orchestrated survey of methodologies for automated

software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

[4] Lionel C Briand, Yvan Labiche, and Zaheer Bawar. 2008. Using machine learning to refine black-box test specifications and test suites. In *2008 The Eighth International Conference on Quality Software*. IEEE, 135–144.

[5] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.

[6] Cristian Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S Pasareanu, Koushik Sen, Nikolai Tillmann, and Willem Visser. 2011. Symbolic execution for software testing in practice: preliminary assessment. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 1066–1071.

[7] Nour Chetouane, Franz Wotawa, Hermann Felbinger, and Mihai Nica. 2020. On Using k-means Clustering for Test Suite Reduction. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 380–385.

[8] Sai Charan Raj Chitirala. 2015. Comparing the effectiveness of automated test generation tools" EVOSUITE" and" Tpalus". (2015).

[9] Ermira Daka, José Campos, Gordon Fraser, Jonathan Dorn, and Westley Weimer. 2015. Modeling readability to improve unit tests. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 107–118.

[10] Javier Ferrer, Francisco Chicano, and Enrique Alba. 2013. Estimating software testing complexity. *Information and Software Technology* 55, 12 (2013), 2125–2139.

[11] 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.

[12] Wael H Gomaa, Aly A Fahmy, et al. 2013. A survey of text similarity approaches. *International Journal of Computer Applications* 68, 13 (2013), 13–18.

[13] Giovanni Grano, Timofey V Titov, Sebastiano Panichella, and Harald C Gall. 2018. How high will it be? using machine learning models to predict branch coverage in automated testing. In *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 19–24.

[14] M Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2, 3 (1993), 270–285.

[15] Hojun Jaygarl, Kai-Shin Lu, and Carl K Chang. 2010. GenRed: A tool for generating and reducing object-oriented test cases. In *2010 IEEE 34th Annual Computer Software and Applications Conference*. IEEE, 127–136.

[16] Saif Ur Rehman Khan, Sai Peck Lee, Raja Wasim Ahmad, Adnan Akhunzada, and Victor Chang. 2016. A survey on Test Suite Reduction frameworks and tools. *International Journal of Information Management* 36, 6 (2016), 963–975.

[17] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[18] Danilo Mandic and Jonathon Chambers. 2001. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley.

[19] 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*. 815–816.

[20] Sankar K Pal and Sushmita Mitra. 1992. Multilayer perceptron, fuzzy sets, classifiaction. (1992).

[21] Annibale Panichella and Urko Rueda Molina. 2017. Java unit testing tool competition-fifth round. In *2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 32–38.

[22] Abdelilah Sakti, Gilles Pesant, and Yann-Gaël Guéhéneuc. 2017. JTeXpert at the SBST 2017 tool competition. In *2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 43–46.

[23] Alex J Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and computing* 14, 3 (2004), 199–222.

[24] Yan-Yan Song and LU Ying. 2015. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry* 27, 2 (2015), 130.

[25] Qiang Sun, Wen-Xin Zhou, and Jianqing Fan. 2020. Adaptive huber regression. *J. Amer. Statist. Assoc.* 115, 529 (2020), 254–265.

[26] Ramsay Taylor, Mathew Hall, Kirill Bogdanov, and John Derrick. 2012. Using behaviour inference to optimise regression test sets. In *IFIP International Conference on Testing Software and Systems*. Springer, 184–199.

[27] Rand R Wilcox. 2004. Some results on extensions and modifications of the Theil-Sen regression estimator. *Brit. J. Math. Statist. Psych.* 57, 2 (2004), 265–280.

[28] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[29] Fangke Ye, Shengtian Zhou, Anand Venkat, Ryan Marucs, Nesime Tatbul, Jesmin Jahan Tithi, Paul Petersen, Timothy Mattson, Tim Kraska, Pradeep Dubey, et al. 2020. MISIM: An End-to-End Neural Code Similarity System. *arXiv preprint arXiv:2006.05265* (2020).

[30] Sai Zhang, David Saff, Yingyi Bu, and Michael D Ernst. 2011. Combined static and dynamic automated test generation. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. 353–363.

[31] Gang Zhao and Jeff Huang. 2018. Deepsim: deep learning code functional similarity. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 141–151.