Static Test Flakiness Prediction

Valeria Pontillo
SeSa Lab — Department of Computer Science, University of Salerno
Fisciano, Italy
vpontillo@unisa.it

ABSTRACT

The problem of flakiness occurs when a test case is non-deterministic and exhibits both a passing and failing behavior when run against the same code. Over the last years, the software engineering research community has been working toward defining approaches for detecting and addressing test flakiness, but most of these approaches suffer from scalability issues. Recently, this limitation has been targeted through machine learning solutions that could predict flaky tests using various features, both static and dynamic. Unfortunately, the proposed solutions involve features that could be costly to compute. In this paper, I perform a step forward and predict test flakiness *only using statically computable metrics.* I conducted an experiment on 18 JAVA projects coming from the FLAKEFLAGGER dataset. First, I statistically assess the differences between flaky and non-flaky tests in terms of 25 static metrics in an individual and combined way. Then, I experimented with a machine learning approach that predicts flakiness based on the previously evaluated factors. The results show that static features can be used to characterize flaky tests: this is especially true for metrics and smells connected to source code complexity. In addition, this new static approach has performance comparable to the machine learning models already in the literature in terms of F-Measure.

ACM Reference Format:

Valeria Pontillo. 2022. Static Test Flakiness Prediction. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3510454.3522680

1 INTRODUCTION AND MOTIVATION

During regression testing, the term *flaky test* is used to describe a test case that shows a non-deterministic behavior when run against the same code. According to the literature, this unexpected behavior does not only make the test result unreliable but (1) may hide real defects and be hard to reproduce [19]; (2) increase testing costs, as developers, invest time debugging failures that are not real [16]; (3) can reduce the overall developer's confidence on test cases, potentially leading to neglect real defects [9]. The consequences of test flakiness have been made more and more popular by practitioners and companies worldwide (e.g., [10, 21]), who all called for automated mechanisms to detect and predict them. The software engineering research community has been contributing with empirical investigations aiming at eliciting the causes of flakiness [9, 17–20] as well as with the definition of techniques for detecting and addressing them [4, 8, 29, 31]. Despite the efforts made by the researchers, the proposed solutions for both detection and prediction of flakiness are not always optimal since these solutions suffer from poor scalability or involve features that are costly to calculate. It seems necessary to identify alternative techniques for detecting and predicting flaky tests.

2 RELATED WORK

The discussion concerns only the seminal papers on the topic that have inspired this work. From the perspective of flakiness detection, researchers devised alternatives like DEFLAKER [4], that works at commit-level and relies on the differential code coverage extracted from the analysis of test execution from a commit to another. Moreover, the use of machine learning approaches has been proposed to predict the presence of a flaky tests. Pinto et al. [24] and further replications [6, 12] exploited the test code dictionary to discriminate the presence of potential flakiness. More recently, Alshammari et al. [1] devised a supervised learning model that, using a mixture of code and coverage metrics, can predict flaky tests with an accuracy up to 86%. While these previous research efforts have shown promising results, they all involve steps that might deteriorate the scalability of the proposed techniques. More particularly, the techniques proposed by Bell et al. [4], and Alshammari et al. [1] require the computation of dynamic features, while the approach by Pinto et al. [24] relies on natural language processing, which is known to be costly as the corpus of the text to analyze increases in size [3]. Recently Pontillo et al. [25] aimed at conducting a feasibility study to assess whether a static prediction of test flakiness would be possible, i.e., whether we could identify likely flaky test cases only based on their design. In particular, the authors analyzed the IDFLAKIES dataset,[1] and investigated the differences between flaky and non-flaky tests in terms of 25 test and production code metrics and smells. The results achieved by this work indicated the feasibility of devising a static approach to flaky tests prediction.

3 PROPOSED SOLUTION

I replicated the work proposed by Pontillo et al. [25] on the FLAKEFLAGGER dataset,[2] in order to increase the generalizability of the results. This analysis was conducted on a new dataset of 9,785 test cases, including 670 flaky tests. After this initial replication, which showed statistically significant differences between flaky and non-flaky sets for metrics connected to code complexity and assertion, I built on top of the replication to devise a prediction model that

[1] The IDFLAKIES dataset: https://sites.google.com/view/flakytestdataset/home.
[2] The FLAKEFLAGGER dataset: https://zenodo.org/record/4450723#.YXetWprP2Uk.

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05…$15.00
https://doi.org/10.1145/3510454.3522680

could identify flaky tests only considering the design of test cases. For this study, I evaluated Decision Trees [11], Naive Bayes [30], Multilayer Perceptron [28], and Support Vector Machine [23] as basic classifiers. Additionally, I also considered two ensemble techniques such as Ada Boost [27], and Random Forest [13]. In terms of training, I had to deal with the flaky test problem being unbalanced. The number of flaky test instances represented the 6.8% of the total amount of test cases in the dataset. Hence, before running the models, I applied the *Synthetic Minority Oversampling Technique*, a.k.a SMOTE [7], to balance the data. I employed a ten-fold cross-validation [5, 14], applying it on both individual projects and considering all projects as a unique dataset. The first necessary step is related to the feature engineering process, that is, the identification of the relevant metrics to use as predictors. While the statistical exercise conducted as the first step provided indications on which features are more connected to test flakiness, it does not necessarily provide insights into the predictive power of the considered metrics [2]. I was interested in assessing the value of the metrics as features of a machine learner more precisely. Hence, I performed a further step ahead by quantifying the predictive power of each metric in terms of information gain [26].

Table 1: List of features with information gain (IG). I have chosen to include features with IG>0.

FlakeFlagger dataset			
Features	**IG**	**Features**	**IG**
LOC	0.1267	TLOC	0.0253
Halstead Vocabulary	0.1239	Complex Class	0.0188
Halstead Length	0.1117	McCabe	0.0178
LCOM2	0.1063	Mystery Guest	0.0152
WMC	0.0998	Assertion Roulette	0.0101
MPC	0.0978	Conditional Test Logic	0.0061
RFC	0.0787	Eager Test	0.0061
Halstead Volume	0.0558	Fire and Forget	0.0021
Spaghetti Code	0.0355	Functional Decomposition	0.0017
CBO	0.0305	Assertion Density	0.0015

4 PRELIMINARY RESULTS

To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the `nemenyi` function available in `R` toolkit,[3] have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are *Eager Test*, *Mystery Guest*, and *Spaghetti Code*, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifiers for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

FlakeFlagger			Random Forest						
Project	**Tests**	**Flaky Tests**	**TP**	**TN**	**FP**	**FN**	**Pr**	**R**	**F**
achilles	1,053	4	2	1,048	1	2	66%	50%	57%
activiti	169	16	5	139	14	11	26%	31%	28%
alluxio	186	122	122	62	2	0	98%	100%	99%
ambari	294	52	47	242	0	5	90%	94%	99%
elastic-job-lite	521	3	2	515	3	1	40%	66%	50%
hbase	368	121	109	233	14	12	88%	90%	89%
hector	121	33	25	81	7	8	78%	75%	76%
httpcore	524	15	9	503	6	6	60%	60%	60%
http-request	161	18	0	143	0	18	NaN	0%	NaN
incubator-dubbo	1,681	18	3	1,649	14	15	17%	16%	17%
java-websocket	107	21	20	85	1	1	95%	95%	95%
logback	655	15	2	639	1	13	66%	13%	22%
ninja	352	16	16	331	5	0	76%	100%	86%
okhttp	782	108	51	609	65	57	43%	47%	45%
orbit	26	4	2	20	2	2	50%	50%	50%
spring-boot	1,634	82	63	1,535	17	19	78%	76%	77%
undertow	48	6	2	39	3	4	40%	33%	26%
wro4j	1,103	16	0	1,081	6	16	0%	0%	NaN
Total	9,785	670	473	8,980	135	197	77%	70%	74%

Table 2 reports the results obtained with the Random Forest classifier. We can observe that there are only two projects where the number of true positives was zero, i.e., HTTP-REQUEST and WRO4J. Besides these two cases, we could observe that the performance is almost always good, except for five projects in which the F-Measure does not even reach 50%. When putting all projects together, the number of true positives was high (473), and the number of false positives was low (135), with the performance metrics ranging from 70% to 77%. My results provide two main insights. First, a fully static approach could reach high levels of precision in situations where the number of flaky tests is large enough or their diversity is low enough to ensure the learning of their characteristics. Second, there are projects for which the use of machine learning does not look reasonable: further research should be done to investigate when to use machine learning or complement it with heuristic approaches that could assist when learning is not a suitable option.

5 CONTRIBUTIONS AND FUTURE WORK

I presented a new approach to predict flaky tests based on statically computable metrics that have been previously analyzed. This empirical study has shown that (1) features related to code complexity metrics and test smells are the metrics that contribute most to this static approach; (2) the newly devised machine learning model achieves performance up to 74% in terms of F-Measure, being no much worse than techniques that adopt more complex and/or dynamic computations. My future work will be focused on understanding the relation between statically computable factors (e.g., code complexity or test smells) and test flakiness. Finally, I aim at conducting additional investigations on how to best configure machine learning pipelines for the problem of flaky test prediction.

[3] https://www.r-project.org/

REFERENCES

[1] A. Alshammari, C. Morris, M. Hilton, and J. Bell. 2021. Flakeflagger: Predicting flakiness without rerunning tests. In *ICSE 2021*. IEEE, 1572–1584.

[2] B Azhagusundari, Antony Selvadoss Thanamani, et al. 2013. Feature selection based on information gain. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 2, 2 (2013), 18–21.

[3] Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*. 26–33.

[4] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, and D. Marinov. 2018. DeFlaker: Automatically detecting flaky tests. In *ICSE 2018*. IEEE, 433–444.

[5] Yoshua Bengio and Yves Grandvalet. 2004. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *J. Mach. Learn. Res.* 5 (Dec. 2004), 1089–1105.

[6] B. Camara, M. Silva, A. Endo, and S. Vergilio. 2021. What is the Vocabulary of Flaky Tests? An Extended Replication. *arXiv preprint arXiv:2103.12670* (2021).

[7] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[8] B. Daniel, V. Jagannath, D. Dig, and D. Marinov. 2009. ReAssert: Suggesting repairs for broken unit tests. In *ASE 2009*. IEEE, 433–444.

[9] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli. 2019. Understanding flaky tests: The developer's perspective. In *ESEC/FSE 2019*. 830–840.

[10] M. Fowler. 2011. Eradicating non-determinism in tests. *Martin Fowler Personal Blog* (2011).

[11] Yoav Freund and Llew Mason. 1999. The alternating decision tree learning algorithm. In *icml*, Vol. 99. Citeseer, 124–133.

[12] G. Haben, S. Habchi, M. Papadakis, M. Cordy, and Y. Le Traon. 2021. A Replication Study on the Usability of Code Vocabulary in Predicting Flaky Tests. In *MSR 2021*.

[13] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[14] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (Montreal, Quebec, Canada) *(IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1137–1143.

[15] Alex J Koning, Philip Hans Franses, Michele Hibon, and Herman O Stekler. 2005. The M3 competition: Statistical tests of the results. *International Journal of Forecasting* 21, 3 (2005), 397–409.

[16] F. Lacoste. 2009. Killing the gatekeeper: Introducing a continuous integration system. In *2009 agile conference*. IEEE, 387–392.

[17] W. Lam, S. Winter, A. Astorga, V. Stodden, and D. Marinov. 2020. Understanding Reproducibility and Characteristics of Flaky Tests Through Test Reruns in Java Projects. In *ISSRE 2020*. IEEE, 403–413.

[18] W. Lam, S. Winter, A. Wei, T. Xie, D. Marinov, and J. Bell. 2020. A large-scale longitudinal study of flaky tests. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–29.

[19] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. 2014. An empirical analysis of flaky tests. In *ESEC/FSE 2014*. 643–653.

[20] A. Memon and M. Cohen. 2013. Automated testing of GUI applications: models, tools, and controlling flakiness. In *ICSE 2013*. IEEE, 1479–1480.

[21] J. Micco. 2017. The state of continuous integration testing@ Google. (2017).

[22] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons*. Princeton University.

[23] William S Noble. 2006. What is a support vector machine? *Nature biotechnology* 24, 12 (2006), 1565–1567.

[24] G. Pinto, B. Miranda, S. Dissanayake, M. D'Amorim, C. Treude, and A. Bertolino. 2020. What is the vocabulary of flaky tests?. In *MSR 2020*. 492–502.

[25] Valeria Pontillo, Fabio Palomba, and Filomena Ferrucci. 2021. Toward Static Test Flakiness Prediction: A Feasibility Study *(MaLTESQuE 2021)*. Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/3472674.3473981

[26] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1, 1 (1986), 81–106.

[27] Robert E Schapire. 2013. Explaining adaboost. In *Empirical inference*. Springer, 37–52.

[28] H Taud and JF Mas. 2018. Multilayer perceptron (MLP). In *Geomatic Approaches for Modeling Land Change Scenarios*. Springer, 451–455.

[29] V. Terragni, P. Salza, and F. Ferrucci. 2020. A container-based infrastructure for fuzzy-driven root causing of flaky tests. In *ICSE 2020*. 69–72.

[30] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. 2010. Naïve Bayes. *Encyclopedia of machine learning* 15 (2010), 713–714.

[31] S. Zhang, D. Jalali, J. Wuttke, K. Muşlu, W. Lam, M. Ernst, and D. Notkin. 2014. Empirically revisiting the test independence assumption. In *ISSTA 2014*. 385–396.