2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 978-1-6654-0337-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASE51524.2021.9678701

Tackling Flaky Tests: Understanding the Problem and Providing Practical Solutions

Martin Gruber
BMW Group and University of Passau
Munich, Germany
Martin.GR.Gruber@bmw.de

***Abstract*—Non-deterministically behaving tests impede software development as they hamper regression testing, destroy trust, and waste resources. This phenomenon, also called test *flakiness*, has received increasing attention over the past years. The multitude of both peer-reviewed literature and online blog articles touching the issue illustrates that flaky tests are deemed both a relevant research topic and a serious problem in everyday business. A major shortcoming of existing work aiming to mitigate test flakiness is its limited applicability, since many of the proposed tools are highly relying on specific ecosystems. This issue also reflects on various attempts to investigate flaky tests: Using mostly similar sets of open-source Java projects, many studies are unable to generalize their findings to projects laying beyond this scope. On top of that, a holistic understanding of test flakiness also suffers from a lack of analyses focusing on the developers' perspective, since most existing studies take a code-centric approach.**

With my work, I want to close these gaps: I plan to create an overarching and widely applicable framework that empowers developers to tackle flaky tests through existing and novel techniques and enables researchers to swiftly deploy and evaluate new approaches. As a starting point, I am studying test flakiness from previously unconsidered angles: I widen the scope of observation investigating flakiness beyond the realm of the Java ecosystem, while also capturing the practitioners' opinion. By adding to the understanding of the phenomenon I not only hope to close existing research gaps, but to retrieve a clear vision of how research on test flakiness can create value for developers working in the field.

***Index Terms*—Flaky Test; Flakiness; Non-determinism**

I. Motivation and State-of-the-Art

Tests that behave non-deterministically hinder software development, especially as they undermine regression testing: Executing tests after every code change exposes regressions earlier and eases the debugging process since incremental testing allows us to infer that the transition of a test case from passing to failing is caused by the most recent code change. This technique is based on the assumption that all tests deliver deterministic outcomes, which is not true for flaky tests, that pass and fail on the same code under test (CUT) due to reasons such as network fluctuations or concurrency related issues.

Having flaky tests in their test suite causes developers to lose trust in testing since the failures they have to investigate are not caused by recent code changes, but either sporadically manifesting bugs or tests making wrong assumptions about the product or the execution environment. Besides eroding trust and causing frustration, flaky tests also increase both human and computational resource costs and delay test feedback.

While these commonly stated effects of flakiness have not been systematically quantified yet, studies showed that even a few flaky tests can cause builds to fail frequently [1] and receive more crash reports [2]. So far, 15 different causes of test flakiness have been identified [3]–[6] with most, but not all, non-deterministic behavior originating from the test code [1], [3], [4]. The majority of flaky tests have been found to be already flaky in the commit in which the test was introduced [3], [7], [8]. Practitioners report flakiness to be far more common among larger and more resource intensive tests [9] and less common in simple and small test cases [8], which might be caused by larger tests using features such as networking and multi-threading [10], that reportedly cause flakiness. Apart from mining software repositories, recent work also investigates human factors [4], [8], however, perceptional analyses remain vastly underrepresented compared to code based studies.

Similar to bugs, flakiness is considered a problem that cannot be eradicated completely [11], [12]. To mitigate the issue, multiple approaches have been proposed that try to detect flaky tests using fewer reruns [13], [14] or find their root causes [15], [16]. Some tools aim at detecting and eradicating certain types of flakiness like order-dependencies [17], [18] and async wait [1], while others try to simulate the impacts of flakiness on other bug fixing techniques [19] showing its disastrous effects on fault localization and patch generation [20]–[22]. A major shortcoming of most existing tools is their limited applicability with many of them relying on specific languages, build-, test-, or instrumentation-frameworks.

II. Planned Contributions

Studying related work, I see two main gaps in existing literature: First, to truly understand the problems caused by flaky tests, we need broader investigations studying flakiness in different technology stacks, from a more developer-focused perspective also considering industrial applications. Second, to truly help developers in dealing with flaky tests, we need to create widely and easily applicable tools which are not limited to a specific ecosystem. In the following sections I discuss both challenges in detail and how I plan to approach them.

A. Understanding Flakiness

When it comes to studying flaky tests, we almost turn a blind eye on the developer's opinion with most studies taking a code-centric approach mining software repositories while not

Work licensed under Creative Commons Attribution 4.0 License. https://creativecommons.org/licenses/by/4.0/

DOI 10.1109/ASE51524.2021.00109

harvesting the insights laying in the perceptional component. This is especially surprising since one of the most agreed upon impacts of flaky tests is losing trust in testing, which is a purely perceptional consequence. Furthermore, there is an imbalance regarding the ecosystems that are being considered, specifically, I see a tendency towards open-source Java projects with proprietary and industrial projects as well as projects using other languages being underrepresented.

To overcome these limitations and better understand the problem of test flakiness, I view it from both a technical angle, looking beyond the scope of previous studies, and a perceptional angle, firstly quantifying key properties of flaky tests using a representative industrial survey. In our recent work [6], we investigated flakiness in Python, a previously untouched area. We searched for flaky tests in public Python repositories measuring their prevalence, causes and degree. While we found flakiness to be equally prevalent in Python as it is in Java, its causes were much different as we encountered more order dependency, network and randomness related flakiness than prior studies. We also discovered infrastructure flakiness as a prominent cause of non-deterministic test outcomes.

Furthermore, we are currently conducting a survey among professional software developers and testers, asking them about their view on test flakiness. Specifically, we want to know more about the causes and consequences of flaky tests, the mitigation strategies that developers currently apply, and the wishes they have towards researchers and tool developers. To paint a holistic picture that captures both the general opinion on test flakiness and the detailed view from within a specific domain, we conduct our study on two groups: One, which is a global sample of professional software developers and testers working in various companies, industries, and countries and one consisting of developers from a large automotive OEM. Through this survey, we hope to provide directions for future research in order to be of value to practitioners in the field.

B. A Guiding Framework

Another shortcoming of existing work on flakiness is that most tools proposed to identify, analyze, and mitigate flaky tests strongly rely on specific build-, test-, or instrumentation-frameworks, thereby highly restricting their applicability. This does not only limit the practical value of these applications, it also hurts the generalizability of the evaluations conducted on them. Zheng et al. [23] support this claim, stating that no unified benchmark has been established to verify the effectiveness of flaky test detection and fixing technologies.

To overcome this limitation, I plan to build a generic framework that provides common interfaces for techniques addressing flaky tests. Such a framework will improve the comparability, generalizability and reproducibility of existing approaches. Furthermore, we hope that it will guide, ease and foster the development of future techniques. Lastly, it will lower the adoption threshold and enable practitioners to benefit from recent advancements in research.

To create a language-agnostic and universally applicable framework, we need to minimize the assumptions we make about the target system as well as the resources and access rights we require. Therefore, I plan to limit the framework to only depend on information that can be retrieved easily without requiring the usage of ecosystem-exclusive features or expensive operations. Examples for such commonly available artifacts are test outcomes, coverage, version control history, source code, or test execution logs.

C. Widely Applicable Detection and Root Causing Techniques

Building on the framework I want to answer two core questions: (1) Given a test case, is it flaky? (2) Given a flaky test, what is the cause of its flakiness? I deem these questions most central since being able to answer them in an accurate, efficient and automated fashion would enable a variety of other, more sophisticated responses to test flakiness such as quantification of flakiness, prioritization of issues, or test quarantining.

1) Detecting Flaky Tests: The prerequisite for any such analysis or action against flaky tests is knowing if a test is actually flaky. While classifying a single test run as flaky or non-flaky might be trivial for a skilled developer, the sheer size of many test suites and number of executions per day make this task infeasible for humans. The most common way to determine if a failing test is flaky or actually broken is to rerun it. However, this is neither always possible (e.g., for long-running tests or expensive hardware tests), nor always suitable, since many causes of flakiness have temporal effects (like network outages) and will still exist in case of an immediate rerun. Therefore, an accurate and efficient method to distinguish flaky tests from non-flaky ones is needed. Existing approaches for flakiness detection either target a specific build system [14], [24] or have not been evaluated thoroughly [25], [26]. With my work, I want to build classifiers that are able to detect flaky tests based on features that can be extracted from existing test executions without requiring additional reruns. The first and most simple step will be trying to predict test flakiness only based on former test outcomes.

2) Root Causing Flaky Tests: Having unveiled flaky tests, in order to fix or mitigate them, developers need insights about the causes of the tests' non-deterministic behavior. Among others, this includes the root cause of the flakiness (i.e. which of the 15 categories it belongs to), its origin (i.e. source code, test code, test infrastructure) and, if applicable, its location inside the code. Unfortunately, there are not many tools aiming at root causing flaky tests and the one that does [15] depends on a C# instrumentation framework limiting it to a niche use case. To develop more widely applicable techniques, we plan to analyze execution traces and logs to find patterns which are indicative for specific types of flakiness that can help to automatically narrow down the causes and locations of flakiness.

III. Evaluation plan

Through my PhD position at BMW, I have the opportunity to quantitatively evaluate the performance of my approaches on large, industrial system and to work with their developers receiving continuous feedback. Nevertheless, I will also be looking at software systems of other origin to evaluate flakiness detection and root causing techniques.

References

[1] W. Lam, K. Muşlu, H. Sajnani, and S. Thummalapenta, "A study on the lifecycle of flaky tests," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1471–1482.

[2] M. T. Rahman and P. C. Rigby, "The impact of failing, flaky, and high failure tests on the number of crash reports associated with firefox builds," in *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2018, pp. 857–862.

[3] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2014, pp. 643–653.

[4] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "Understanding flaky tests: The developer's perspective," in *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2019, pp. 830–840.

[5] S. Thorve, C. Sreshtha, and N. Meng, "An empirical study of flaky tests in android apps," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 2018, pp. 534–538.

[6] M. Gruber, S. Lukasczyk, F. Kroiß, and G. Fraser, "An empirical study of flaky tests in python," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 148–158.

[7] W. Lam, S. Winter, A. Wei, T. Xie, D. Marinov, and J. Bell, "A large-scale longitudinal study of flaky tests," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–29, 2020.

[8] A. Ahmad, O. Leifler, and K. Sandahl, "Empirical analysis of factors and their effect on test flakiness-practitioners' perceptions," *arXiv preprint arXiv:1906.00673*, 2019.

[9] J. Listfield, "Where do our flaky tests come from?" Apr. 2017. [Online]. Available: https://testing.googleblog.com/2017/04/where-do-our-flaky-tests-come-from.html

[10] S. Stewart, "Test sizes," Dec. 2010. [Online]. Available: https://testing.googleblog.com/2010/12/test-sizes.html

[11] J. Raine, "Reducing flaky builds by 18x," Dec. 2020. [Online]. Available: https://github.blog/2020-12-16-reducing-flaky-builds-by-18x/

[12] J. Micco, "Flaky tests at google and how we mitigate them," 2016, accessed: 2020–10–19. [Online]. Available: https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html

[13] W. Lam, R. Oei, A. Shi, D. Marinov, and T. Xie, "Idflakies: A framework for detecting and partially classifying flaky tests," in *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2019, pp. 312–322.

[14] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, and D. Marinov, "Deflaker: Automatically detecting flaky tests," in *International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 433–444.

[15] W. Lam, P. Godefroid, S. Nath, A. Santhiar, and S. Thummalapenta, "Root causing flaky tests in a large-scale industrial setting," in *International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2019, pp. 101–111.

[16] D. C. Celal Ziftci, "De-flake your tests: Automatically locating root causes of flaky tests in code at google," in *ICSME 2020-International Conference on Software Maintenance and Evolution*, 2020. [Online]. Available: https://www.youtube.com/watch?v=uMGWf0tFqjM

[17] A. Gambi, J. Bell, and A. Zeller, "Practical test dependency detection," in *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2018, pp. 1–11.

[18] A. Shi, W. Lam, R. Oei, T. Xie, and D. Marinov, "ifixflakies: A framework for automatically fixing order-dependent flaky tests," in *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2019, pp. 545–555.

[19] M. Cordy, R. Rwemalika, M. Papadakis, and M. Harman, "Flakime: Laboratory-controlled test flakiness impact assessment. A case study on mutation testing and program repair," *CoRR*, vol. abs/1912.03197, 2019.

[20] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1936–1964, 2017.

[21] B. Vancsics, T. Gergely, and Á. Beszédes, "Simulating the effect of test flakiness on fault localization effectiveness," in *2020 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*. IEEE, 2020, pp. 28–35.

[22] Y. Qin, S. Wang, K. Liu, X. Mao, and T. F. Bissyandé, "On the impact of flaky tests in automated program repair," 2021.

[23] W. Zheng, G. Liu, M. Zhang, X. Chen, and W. Zhao, "Research progress of flaky tests," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 639–646.

[24] A. Alshammari, C. Morris, M. Hilton, and J. Bell, "Flakeflagger: Predicting flakiness without rerunning tests," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1572–1584.

[25] S. Liviu, "A machine learning solution for detecting and mitigating flaky tests," Oct. 2019. [Online]. Available: https://medium.com/fitbit-tech-blog/a-machine-learning-solution-for-detecting-and-mitigating-flaky-tests-c5626ca7e853

[26] M. Machalica, W. Chmiel, S. Swierc, and R. Sakevych, "How do you test your tests?" Dec. 2020. [Online]. Available: https://engineering.fb.com/2020/12/10/developer-tools/probabilistic-flakiness/