Flaky Tests: Problems, Solutions, and Challenges

Fabio Palomba
SeSa Lab - University of Salerno, Italy
fpalomba@unisa.it

Test cases represent the first defensive line against the introduction of software faults, especially when developers test for regressions: Unfortunately, however, test cases are not immune to defects. One of the most critical issues affecting tests is called "flakiness" and appears when a test exhibits a seemingly random outcome (*i.e.,* pass or fail) despite exercising code that has not been changed. The presence of flaky tests may cause substantial problems to developers: (1) they may hide read faults, other than being hard to reproduce because of their non-determinism; (2) they increase maintenance costs, as developers might spend additional time to debug failures that are not connected to any fault; and (3) they can reduce the overall developer's confidence in testing.

1 Problems

Luo *et al.* [1] empirically explored the reasons behind flaky tests. They discovered that 45% of the flaky tests considered were due to Async Wait issues: in particular, these tests make asynchronous calls but do not properly wait for the result of these calls. As an example, a test that waits for the response of a remote server using a Thread.sleep statement may be flaky depending on either the milliseconds passed to the sleep method or how fast the server responds. Other common causes relate to Concurrency issues. These results were later confirmed and extended by Eck *et al.* [2], who surveyed developers on the root causes of flaky tests. The authors confirmed the categories of the previous studies, but also identified three new root causes. Furthermore, they discovered that in some cases the flakiness can be due to problems originating in the production code, *i.e.,* changes applied by developers when enhancing source code may eventually lead to disrupt the associated tests and make them flaky.

In: D. Di Nucci, C. De Roover (eds.): Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, 28-11-2019, published at http://ceur-ws.org

2 Solutions

Besides studying the root causes of flaky tests, researchers have been also working on their automatic identification. The simplest solution is given by the ReRun approach: it consists of re-running tests multiple times and checking whether their outcome change at least once. Anecdotal evidence suggests to re-run tests ten times, however no previous study has systematically assessed this aspect. The ReRun approach, however, has a major drawback: scalability. An alternative is represented by DeFlaker [3]: this is a technique based on a mix of static and dynamic analysis that is suitable to be ran within continuous integration pipelines. In particular, starting from a newly committed change available in the repository, DeFlaker runs a differential coverage analysis, with which it identifies the lines of code that have been modified or added to a file F in the commit c. Afterwards, the approach runs test cases and monitors the statement coverage they have on F at both times c_{-1} and c: this step outputs the lines of code that the tests cover on the current and previous version of F. Finally, DeFlaker identifies flaky tests in two cases: (1) if a test passes when ran on F at time c_{-1} but fails when ran at time c and does not cover the lines that have been modified or changed in c; or (2) if a test fails when ran on F at time c_{-1} but passes when ran at time c and does not cover the lines that have been modified or changed in c.

3 Challenges

Eck *et al.* [2] identified a number of critical open issues to deal with flakiness. First and foremost, developers indicated that understanding the context of the failure represents the most critical part of the identification and fixing process. When a test fails, diagnosing it and designing a solution might take long because developers have to understand what led the test to fail. As such, they highlighted flaky test replay approaches as the most desirable.

The second challenge for developers is to understand the root cause of flaky tests: flakiness can arise in

several different manners and a timely identification of the root cause can help developers with both the allocation of resources and the actual identification of the problem. Hence, creating techniques that can automatically classify the root causes of flaky tests represents a priority for the research community.

Finally, it is crucial for developers to quickly understand where to look at when diagnosing flaky tests. This would substantially decrease the time required to fix the flakiness. Thus, this challenge represents a call for researchers, who may either adapt existing fault localization approaches or define specialized methods for locating the root cause of flaky tests.

References

[1] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 643–653. ACM, 2014.

[2] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. Understanding flaky tests: The developer's perspective. 2019.

[3] Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung, and Darko Marinov. Deflaker: automatically detecting flaky tests. In *Proceedings of the 40th International Conference on Software Engineering*, pages 433–444. ACM, 2018.