Regression-Test History Data for Flaky-Test Research

Philipp Wendler
LMU Munich
Germany

Stefan Winter
LMU Munich
Germany

ABSTRACT

Due to their random nature, flaky test failures are difficult to study. Without having observed a test to both pass and fail under the same setup, it is unknown whether a test is flaky and what its failure rate is. Thus, flaky-test research has greatly benefited from data records of previous studies, which provide evidence for flaky test failures and give a rough indication of the failure rates to expect. For assessing the impact of the studied flaky tests on developers' work, it is important to also know how flaky test failures manifest over a regression test history, i.e., under continuous changes to test code or code under test. While existing datasets on flaky tests are mostly based on re-runs on an invariant code base, the actual effects of flaky tests on development can only be assessed across the commits in an evolving commit history, against which (potentially flaky) regression tests are executed. In our presentation, we outline approaches to bridge this gap and report on our experiences following one of them. As a result of this work, we contribute a dataset of flaky test failures across a *simulated* regression test history.

KEYWORDS

Flaky Tests, Regression Testing, Test Result Histories, Dataset

ACM Reference Format:
Philipp Wendler and Stefan Winter. 2024. Regression-Test History Data for Flaky-Test Research. In *2024 International Flaky Tests Workshop 2024 (FTW '24), April 14, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3643656.3643901

1 INTRODUCTION

Flaky tests can both pass and fail without changes to test code or code under test (CUT). This makes them problematic for regression testing, because flaky regression tests can falsely indicate defects in the CUT and thereby block further development and integration.

Flaky test detection approaches typically employ test re-executions on an unchanged code base (see [6] for an overview). While research projects commonly use repetition counts in the hundreds (e.g., [4]) or even tens of thousands [1], for many practitioners long detection latencies from many repetitions are impractical. To this end, repetition counts in the single or lower double digit range are used and identified flaky tests subsequently quarantined (instead of repaired).

This difference causes a gap between academic and practical approaches to cope with flaky tests. It is unclear to which degree the flaky tests that are identified with targeted academic detection approaches overlap with the flaky tests that plague developers most in their daily work. In other words: Flaky tests that are detected with an academic tool may not necessarily be the ones that cause most friction in continuous integration (CI). Closing this gap would help researchers to focus their work (if desired) on the most painful practical manifestations of flaky tests.

However, to assess how a detected flaky test affects CI, the behavior of this test in the regression test history of the project must be known, i.e., along a changing test and CUT code base and not just across a number of re-executions on the same commit. While this information is available in the software projects plagued by flaky tests, it is not part of the flaky test datasets that academic research is often based on. To take a first step in solving this problem, we outline different possibilities for obtaining such a *flaky regression-test history*, i.e., a regression test history that includes flaky tests, and propose a dataset of simulated flaky regression test histories for Maven projects in IDoFT [3].

2 OBTAINING FLAKY REGRESSION-TEST HISTORIES

To the best of our knowledge, only one dataset of regression test histories with flaky tests exists to date. Gruber et al. proposed an approach for practical flaky test detection on the basis of regression test histories and code features [2] and published a dataset of test result histories for 200 tests. As the histories have been obtained from an industry collaborator, the dataset is redacted. Test names, implementations, etc. are missing and their test suites and execution environments are unknown. This makes the dataset suitable for "black-box" assessments that solely rely on the published test features, but its applicability beyond that scenario remains limited.

We have identified several options for obtaining flaky regression-test histories based on different data sources, which fall in two classes. In essence, one can either (1) start from regression test datasets and search for flaky tests or (2) start from flaky test datasets and simulate regression test histories.

Regression Test Datasets. Regminer [7] aims to extract regressions from commit histories. The dataset accompanying the Regminer paper contains projects that are also found in flaky test datasets, e.g., Apache Ambari and Hadoop.

In a study of "orange" CI jobs that intermittently pass or fail (in builds or tests), Lampel et al. [5] present a dataset scraped from Mozilla treeherder. The system provides a dedicated view for intermittent test failures[1], which may be leveraged, but only keeps records for 3 weeks, so that building a dataset from this source requires continuous monitoring.

Flaky Test Datasets. IDoFT [3] provides a comprehensive dataset of flaky tests identified with different detectors. Similar datasets exists for individual research projects.

[1] https://treeherder.mozilla.org/intermittent-failures

FTW '24, April 14, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0558-8/24/04...$15.00
https://doi.org/10.1145/3643656.3643901

Table 1: Dataset summary: Slug (Module) – the Maven project's GitHub slug and module name (if applies), FIC hash – flakiness introducing commit, tests/commit counts, average commits per test (as tests may be introduced/removed), nr. of tests flaky in ≥ 1 commit, nr. of tests consistently failing in ≥ 1 commit, nr. of distinct histories that can be generated from the dataset.

Slug (Module)	FIC Hash	Tests	Commits	Avg. Commits/Test	Flaky Tests	Tests w/ Consistent Fails	Distinct Histories
TooTallNate/Java-WebSocket	822d40f5	146	75	75.0	24	1	2.6×10^9
apereo/java-cas-client (cas-client-core)	5e36559b	157	65	61.7	3	2	1.0×10^7
eclipse-ee4j/tyrus (tests/e2e/standard-config)	ce3b8c76	185	16	16.0	12	0	261
feroult/yawp (yawp-testing/yawp-testing-appengine)	abae1782	1	191	191.0	1	1	8
fluent/fluent-logger-java	5fd46383	19	131	105.6	11	2	8.0×10^{32}
fluent/fluent-logger-java	87e957ae	19	160	122.4	11	3	2.1×10^{31}
javadelight/delight-nashorn-sandbox	d0d651ff	81	113	100.6	2	5	4.2×10^{10}
javadelight/delight-nashorn-sandbox	d19eeeab	81	93	83.5	1	5	2.6×10^9
sonatype-nexus-community/nexus-repository-helm	5517c8e6	18	32	32.0	0	0	18
spotify/helios (helios-services)	0232600b	190	448	448.0	0	37	190
spotify/helios (helios-testing)	78a86465	43	474	474.0	0	7	43

Dataset Construction. We construct a dataset of *possible* flaky regression-test histories based on IDoFT, as it contains information for "flakiness-introducing commits" (FICs) and commits on which tests have originally been identified as flaky ("iDFlakies commit" [4]) for a subset of flaky tests in the dataset. These commits can conveniently serve as boundaries for the test result histories we construct. Specifically, we

- select projects from IDoFT with tests that have a known FIC,
- filter out projects, for which we are not able to build and test the Maven modules in the IDoFT dataset in the iDFlakies commit (e.g., due to broken dependencies),
- run the modules' test suites on each commit in the commit history between the FIC and the iDFlakies commit, and
- repeat the test execution across these commits 30 times to increase the likelihood of observing flakiness and increasing the dataset: With 30 repetitions across n commits for each module, we obtain 30^n *possible* regression test histories.

3 INITIAL RESULTS AND FUTURE WORK

So far, we have successfully applied the outlined dataset construction methodology to 11 module/FIC combinations across 8 Maven projects. The dataset[2] contains 28 200 test result histories for 840 tests with history lengths ranging from 1 to 474 commits. The key takeaways from our initial assessment of the dataset are:

Result history variations across 30 repetitions exist, but they are limited. On hindsight, this is an intuitive result. If test results were changing at a high rate across repetitions, the developers would have likely addressed the problem (e.g., by test exclusion).

Using richer result encodings than binary pass/fail yields different results. We observe 5 cases of deviating classifications for binary (pass/fail) and non-binary (result type, exception type, exception message) result encodings in our dataset. In all of these cases, a test in the dataset fails consistently across 30 repetitions on some commit. However, it fails with different exceptions (3 cases) or the same exception but different messages (2 cases). From a manual investigation, these tests are flaky, but do not become visible as such if a binary result encoding is applied, as it then consistently fails across all repetitions. The following observations are based on the non-binary encoding.

Simulated regression test histories mix flaky failures with regression failures. We find 57 tests in 7 modules that fail consistently across all 30 repetitions in one or more commits, out of which 4 tests exhibit both consistent and flaky failures. This makes the dataset a good starting point for assessing the discriminative power of flaky test classifiers like FLAST [8] or FlakeFlagger [1] across flaky- and consistent-failing tests/commits.

Failure distributions vary across commits. Out of 58 simulated flaky test histories that fail on more than a single commit in our dataset, 31 show significantly (Fisher's exact test, $\alpha = 0.05$, 12 p-values via Monte-Carlo simulation) differing failure distributions across commits. This may hint at an opportunity for better debugging of flaky tests, for which higher failure rates are favorable. However, this is an early-stage result based on 30 repetitions and requires further investigation on how commits with higher failure rates can be identified without numerous re-runs.

ACKNOWLEDGMENTS

We appreciate Qingyu Li's help in the data collection process. The authors are supported by the Deutsche Forschungsgemeinschaft (DFG) – 496588242 (IDEFIX) and the LMU Postdoc Support Fund.

REFERENCES

[1] Abdulrahman Alshammari, Christopher Morris, Michael Hilton, and Jonathan Bell. 2021. FlakeFlagger: Predicting Flakiness Without Rerunning Tests. In *Proc. ICSE*. IEEE, 1572–1584. https://doi.org/10.1109/ICSE43902.2021.00140

[2] Martin Gruber, Michael Heine, Norbert Oster, Michael Philippsen, and Gordon Fraser. 2023. Practical Flaky Test Prediction using Common Code Evolution and Test History Data. In *Proc. ICST*. IEEE, 210–221. https://doi.org/10.1109/ICST57152.2023.00028

[3] Wing Lam. 2020. International Dataset of Flaky Tests (IDoFT). http://mir.cs.illinois.edu/flakytests Accessed: 2023-12-07.

[4] Wing Lam, Stefan Winter, Anjiang Wei, Tao Xie, Darko Marinov, and Jonathan Bell. 2020. A Large-Scale Longitudinal Study of Flaky Tests. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 202 (2020), 29 pages. https://doi.org/10.1145/3428270

[5] Johannes Lampel, Sascha Just, Sven Apel, and Andreas Zeller. 2021. When Life Gives You Oranges: Detecting and Diagnosing Intermittent Job Failures at Mozilla. In *Proc. ESEC/FSE*. ACM, 1381–1392. https://doi.org/10.1145/3468264.3473931

[6] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. 2021. A Survey of Flaky Tests. *ACM Trans. Softw. Eng. Methodol.* 31, 1, Article 17 (2021), 74 pages. https://doi.org/10.1145/3476105

[7] Xuezhi Song, Yun Lin, Siang Hwee Ng, Yijian Wu, Xin Peng, Jin Song Dong, and Hong Mei. 2022. RegMiner: Towards Constructing a Large Regression Dataset from Code Evolution History. In *Proc. ISSTA*. ACM, 314–326. https://doi.org/10.1145/3533767.3534224

[8] Roberto Verdecchia, Emilio Cruciani, Breno Miranda, and Antonia Bertolino. 2021. Know You Neighbor: Fast Static Prediction of Test Flakiness. *IEEE Access* 9 (2021), 76119–76134. https://doi.org/10.1109/ACCESS.2021.3082424

[2] https://doi.org/10.5281/zenodo.10639030