A Preliminary Study of Fixed Flaky Tests in Rust Projects on GitHub

Tom Schroeder
University of Illinois Urbana-Champaign
Urbana, IL, USA
jts13@illinois.edu

Minh Phan
University of Illinois Urbana-Champaign
Urbana, IL, USA
minhnp2@illinois.edu

Yang Chen
University of Illinois Urbana-Champaign
Urbana, IL, USA
yangc9@illinois.edu

Abstract—**Prior research has extensively studied flaky tests in various domains, such as web applications, mobile applications, and other open-source projects in a range of multiple programming languages, including Java, JavaScript, Python, Ruby, and more. However, little attention has been given to flaky tests in Rust—an emerging popular language known for its safety features relative to C/C++. Rust incorporates interesting features that make it easy to detect some flaky tests, e.g., the Rust standard library randomizes the order of elements in hash tables, effectively exposing implementation-dependent flakiness. However, Rust still has several sources of nondeterminism that can lead to flaky tests.**

We present our work-in-progress on studying flaky tests in Rust projects on GitHub. Searching through the closed GitHub issues and pull requests, we identified 1,146 issues potentially related to Rust flaky tests. We focus on flaky tests that are fixed, not just reported, as the fixes can offer valuable information on root causes, manifestation characteristics, and strategies of fixes. By far, we have inspected 53 tests. Our initial findings indicate that the predominant root causes include asynchronous wait (33.9%), concurrency issues (24.5%), logic errors (9.4%), and network-related problems (9.4%). Our artifact is publicly available at [1].

Index Terms—**Flaky Tests, Rust**

I. INTRODUCTION

Regression testing is crucial to ensure software reliability by detecting potential bugs after code changes. However, flaky tests can non-deterministically pass or fail when run on the same code version, significantly impacting the quality of regression test suites [2]. Common flaky tests can be categorized into order-dependent (OD) and non-order-dependent (NOD) tests. OD tests can be flaky due to dependencies within the test suite—passing if run before specific tests and failing if run after. NOD tests, on the other hand, may become flaky for reasons other than order dependencies, such as concurrency, asynchronous waits, network, and I/O operations. Implementation-dependent (ID) tests are a subcategory of NOD caused by wrong assumptions of unordered collections in the implementation of tests.

Since the seminal study of flaky tests in Apache projects done by Luo et al. in 2014 [2], several studies have been proposed for characterizing [3]–[13], detecting and fixing flaky tests across various programming languages and platforms, including Android, Java, JavaScript, Python, and Ruby. However, no prior work has specifically focused on flaky tests in Rust projects. Given Rust's emerging popularity and its interesting features that could introduce non-determinism, investigating flakiness in Rust projects is a crucial area for future research.

In this study, we investigate test flakiness in Rust projects. Starting by constructing a dataset of flaky tests from Rust projects on GitHub, we identified 1,146 issues potentially related to Rust flaky tests. By further inspecting the characteristics of 53 tests, we identified nine common root causes of test flakiness. To our knowledge, we presented the first study to investigate flaky tests in Rust on Github projects. Our findings offer valuable insights that could guide future research.

II. METHODOLOGY

We performed a two-step methodology—filtering and investigation—to construct the Rust flaky test dataset.

A. Filtering

To accurately investigate the root causes of flakiness, we focus on *already fixed* flaky tests. Our approach involves leveraging the GitHub REST API to search for issues in repositories using Rust, specifically querying for the term *'flaky'*. The search was narrowed to only closed issues that have linked pull requests. Finally, we create a dataset containing all issues captured on October 29, 2024. This process resulted in a dataset of 1,146 issues potentially related to Rust flaky tests.

B. Investigation

Manually investigating over 1,000 issues is time-consuming. To ensure the diversity of tests in our study within a limited time, we shuffled the dataset and performed our analysis from start to finish. At the time of submission, we have manually investigated a subset of 53 tests from 49 projects of their root causes and fix strategies to conduct a deep analysis through the following process: We first reviewed the description provided in each issue, which may include the test failure scenario, the assertion failure, and the hypothesized source of the issue. We then reviewed the linked pull request to verify it as the fix for flakiness. Finally, we categorized the cause of flakiness and its fixing strategy. Note that we had to exclude certain cases because (1) some issues identified in our GitHub search using 'flaky' or its variations were actually not relevant to flaky tests, and (2) the issues pertained to parts of the repository not utilizing Rust, such as Python bindings or CI infrastructure.
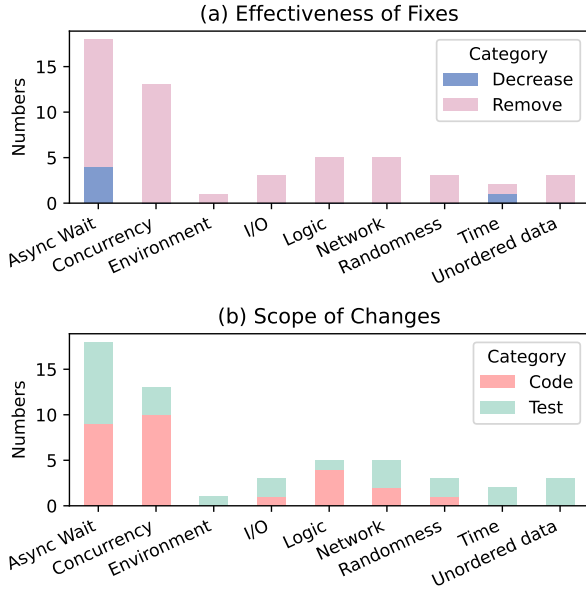
Fig. 1: Categories of fixes. (a) shows the effectiveness of fixes per root cause category; (b) outlines the scope of changes for fixes per root cause category.

III. ANALYSIS AND FUTURE WORK

A. Categories of Flakiness Root Causes

Table I presents the categories identified for the 53 tests through manual analysis in our study. We found that 33.9% of these tests are affected by asynchronous waits. Among these, 38% specifically are related to issues of *improper wait*, which means tests do not properly wait for asynchronous operations to complete before moving on to assertions. Flakiness due to concurrency, logic, and network issues also appear frequently among the top causes. Additional root causes include I/O operations, randomness, time issues, unordered data and environment.

TABLE I: Root causes of flakiness

Root Cause (Percent)	Total	Root Cause (Percent)	Total
Async Wait (33.9%)	18	**Concurrency (24.5%)**	13
Improper Wait	8	Robustness	4
Tweak Duration	2	Retry	2
Sleep	2	Lock	1
Increase Timeout	1	Channel	1
Retry	1	Race Condition	1
Configuration	1	Non-deterministic Environment	1
Deadlock	1	Reorder Initialization	1
Channel	1	Shared State	1
Unknown State	1	Atomics	1
Logic (9.4%)	5	**Network (9.4%)**	5
Off by One	2	Certs	1
Inverted Conditional	1	Connection Loss	1
Deserialization	1	Empty Header	1
Sorting	1	Retry	1
		Reused Resource (port)	1
I/O (5.7%)	3	**Randomness (5.7%)**	3
Flush	1	Ranges	1
Temp Files	1	RNG	1
Resource Limit	1	Weighting	1
Time (3.8%)	2	**Unordered data (5.7%)**	3
Inconsistent Clock	1	Sorting	2
Off by One	1	Non-Deterministic Environment	1
Environment (1.9%)	1		
Shared libraries	1		

B. Categories of Fix Strategies

To investigate the fixes of flaky tests, we categorize them based on the *effectiveness of fixes* by assessing whether each fix completely removes flakiness or only reduces the likelihood of failure (Figure 1(a)). Additionally, we evaluate the *scope of changes*, determining whether the fix modifies test code or main code (Figure 1(b)). Four tests affected by async waits and one test caused by time issues have not been completely fixed but only have a reduced likelihood of test failures. The potential reason could be that the actual root cause was not fully understood or that completely removing flakiness through a patch is programmatically challenging. Additionally, of the 53 tests, 27 were fixed by modifying the main code to address common root causes such as async waits, concurrency, and logic. The remaining tests were patched within the test code.

IV. FUTURE WORK

We present the first study of flaky tests in Rust from GitHub projects. In future work, we aim to continue our study of the dataset with 1,146 issues related to potential Rust flaky tests and explore methods for detecting and addressing flakiness in Rust. Additionally, we will investigate how language features influence flaky tests across different programming languages.

REFERENCES

[1] "Artifact," https://github.com/LALAYANG/RustFlakyTest, 2024.
[2] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 2014, pp. 643–653.
[3] S. Thorve, C. Sreshtha, and N. Meng, "An empirical study of flaky tests in android apps," in *2018 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2018, pp. 534–538.
[4] 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*. IEEE, 2021, pp. 148–158.
[5] M. Gruber, M. F. Roslan, O. Parry, F. Scharnböck, P. McMinn, and G. Fraser, "Do automatic test generation tools generate flaky tests?" in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, 2024.
[6] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, "A survey of flaky tests," *ACM Trans. Softw. Eng. Methodol.*, 2021.
[7] W. Lam, P. Godefroid, S. Nath, A. Santhiar, and S. Thummalapenta, "Root causing flaky tests in a large-scale industrial setting," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 101–111.
[8] 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.
[9] 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, pp. 1–29, 2020.
[10] W. Lam, S. Winter, A. Astorga, V. Stodden, and D. Marinov, "Understanding reproducibility and characteristics of flaky tests through test reruns in Java projects," in *2020 IEEE 31st International Symposium on Software Reliability Engineering*. IEEE, 2020, pp. 403–413.
[11] Y. Chen, A. Yildiz, D. Marinov, and R. Jabbarvand, "Transforming test suites into croissants," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1080–1092.
[12] N. Hashemi, A. Tahir, and S. Rasheed, "An empirical study of flaky tests in javascript," in *2022 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2022, pp. 24–34.
[13] K. Barbosa, R. Ferreira, G. Pinto, M. d'Amorim, and B. Miranda, "Test flakiness across programming languages," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2039–2052, 2022.