Flaky Tests in a Large Industrial Database Management System: An Empirical Study of Fixed Issue Reports for SAP HANA

Alexander Berndt*
alexander.berndt@uni-heidelberg.de
Heidelberg University
Germany

Thomas Bach
thomas.bach03@sap.com
SAP
Germany

Sebastian Baltes
sebastian.baltes@uni-heidelberg.de
Heidelberg University
Germany

Abstract

Flaky tests yield different results when executed multiple times for the same version of the source code. Thus, they provide an ambiguous signal about the quality of the code and interfere with the automated assessment of code changes. While a variety of factors can cause test flakiness, approaches to fix flaky tests are typically tailored to address specific causes. However, the prevalent root causes of flaky tests can vary depending on the programming language, application domain, or size of the software project. Since manually labeling flaky tests is time-consuming and tedious, this work proposes an LLMs-as-annotators approach that leverages intra- and inter-model consistency to label issue reports related to fixed flakiness issues with the relevant root cause category. This allows us to gain an overview of prevalent flakiness categories in the issue reports. We evaluated our labeling approach in the context of SAP HANA, a large industrial database management system. Our results suggest that SAP HANA's tests most commonly suffer from issues related to concurrency (23%, 130 of 559 analyzed issue reports). Moreover, our results suggest that different test types face different flakiness challenges. Therefore, we encourage future research on flakiness mitigation to consider evaluating the generalizability of proposed approaches across different test types.

CCS Concepts

• **Software and its engineering → Empirical software validation**; *Software evolution*; *Development frameworks and environments*; **Software reliability**.

Keywords

Test Flakiness, Software Testing, Empirical Study, Database Management Systems, Large Language Models, Artificial Intelligence

ACM Reference Format:
Alexander Berndt, Thomas Bach, and Sebastian Baltes. 2026. Flaky Tests in a Large Industrial Database Management System: An Empirical Study of Fixed Issue Reports for SAP HANA. In *3rd International Flaky Tests Workshop 2026 (FTW '26), April 12–18, 2026, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3786178.3788327

*Also with SAP.

This work is licensed under a Creative Commons Attribution 4.0 International License.
FTW '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2409-1/2026/04
https://doi.org/10.1145/3786178.3788327

1 Introduction

Flaky tests yield different results when executed multiple times for the same version of the source code. Thus, flaky tests affect various aspects of test automation and represent a major problem for industrial software testing [6, 12, 15, 17]. They hinder automatic merging of code changes, reduce the effectiveness of test prioritization techniques, and erode developers' trust in test results [6, 10, 12].

Previous research has studied the root causes of flakiness and proposed strategies to detect and fix flaky tests [5, 7, 9, 11, 14, 21–23, 29]. Due to the range of potential issues that can cause flakiness, there seems to be no one-size-fits-all solution that can be applied to detect or fix all types of flakiness. Therefore, specialized solutions have emerged that are often tailored to specific root causes of flakiness [8, 21, 35, 41]. Hence, before assessing existing approaches to fix test flakiness, it is beneficial to know the root causes of flaky tests. Although previous studies identified prevalent root causes of flaky failures [22, 29, 36, 37], prior research also suggests that the root causes of flakiness vary between different programming languages and types of software [12, 37]. Therefore, results from previous studies cannot always be generalized to very large software projects that pose unique challenges [2, 25].

Therefore, we performed an empirical analysis of flakiness-related issue reports to gain an overview of the root causes of flaky tests in the context of SAP HANA. SAP HANA is a large industrial database management system that serves as the foundation for SAP's business applications [24]. The SAP HANA test suite consists primarily of two test types: native C++ unit tests and Python-based system tests that interact with a running database instance via SQL. To gain insights into differences in flakiness between test types, we categorized issue reports by affected test type using large language models (LLMs) as annotators. Based on the labeled issue reports, we identified Concurrency as the most prevalent root cause of flakiness in the given dataset, appearing in 130 of 559 issue reports (23%). Based on a comparison with a manually labeled sample of $n = 50$ issue reports, our results indicate that LLMs can enable large-scale flakiness analyses in industrial contexts. Our investigation of cases where manual labels do not match those generated by LLMs shows that some flakiness categories can overlap within a single test. This finding leads to the conclusion that identifying root causes of flaky tests is a multi-label task.

In summary, this work provides the following contributions:

(1) The evaluation of an LLM-as-annotators approach for a large-scale flakiness analysis.
(2) A comparison of the root causes of flakiness between different test types in a very large software project.
(3) An overview of how the numbers of flakiness issue reports evolve over time in a large industrial environment.

Table 1: Flakiness categories derived from related work.

Category	Description
Concurrency	Test failures due to timing issues, thread handling problems, deadlocks, or synchronization issues in multi-threaded or distributed environments [22].
Timeout	Tests that exceed time limits, e.g., due to system load or slow hardware, causing flaky failures [6].
Oracle-Brittleness	Assumptions about ordering, exact error messages, or non-deterministic checks; lack of robustness against minor behavioral differences [28].
Configuration	Incorrect or missing configuration parameters that are not aligned with test expectations [31].
Async Wait	Test failures due to asynchronous waits for external resources, which are not handled robustly [22].
Isolation	Inter-test dependencies, e.g., where tests lack proper cleanup and interfere with other tests, causing cascading failures [29].
Platform	Differences related to processor architecture, compiler behavior, or alignment issues [9].
Application	Failures due to defects in the tested application [32].
Test-Framework	Issues with testing infrastructure, including test harnesses, assertion frameworks, or utilities [26, 31, 32].
Error-Handling	Missing or improper handling of exceptions leading to unexpected behavior [11].
Memory-Management	Failures related to improper memory handling, e.g., use of uninitialized variables or use-after-free [12, 33].
Environment	Test failures, e.g., due to slow or constrained filesystems, that cause flaky failures [40].
Randomization	Tests relying on random behavior without deterministic sampling, leading to flaky outcomes [9].
Network	Transient network errors, connection resets, or flaky inter-service communication [22].
Numeric-Semantics	Numeric inaccuracies caused by limited floating-point precision causing flaky failures [9].
Locale	Timezone differences affecting test expectations [9].

The remainder of this document is structured as follows. We start by introducing the relevant context in Section 2 and Section 3, followed by a description of our dataset in Section 4. In Section 5, we motivate our research questions and the respective methodology before presenting our results in Section 6. Finally, we discuss our findings in Section 7 and conclude this work in Section 8.

2 Related Work

In Table 1, we describe the flakiness categories we used to categorize the issue reports in this work. In the following, we introduce related work on categorizing flaky tests. Note that we adapted the naming of the root cause categories to the language used by SAP developers, as described in the following.

Luo et al. [22] conducted an analysis of flaky tests using 201 flakiness-fixing commits from 51 open-source repositories, mostly written in Java. Based on the examined commits, they identified Async wait, Concurrency, and 'test order dependency' (Isolation in our taxonomy) as the most common root causes for flakiness. Eck et al. [9] also found that Async wait and Concurrency are the most common root causes based on an analysis of 200 flaky tests reported in Mozilla's issue tracker. In addition to Luo et al. [22], they identified 'too restrictive range' (Oracle Brittleness) as the third common category, appearing in 40 of 234 cases (17%).

Romano et al. [36] adopted the methodology introduced by Luo et al. [22] to gain insights into the flakiness of user interface (UI) tests. Based on an analysis of commit messages, bug reports, and code changes for 235 flaky UI tests in 62 open-source projects, they found Async wait to be the most prominent category. However, in contrast to Luo et al. [22], issues related to the execution environment and the test runner for UI tests were the second and third most common root causes of flakiness. This difference suggests that the root causes of flakiness vary between test targets, which developers also reported in existing surveys [9, 12]. Furthermore, results from a developer survey suggested that software for different domains may also suffer from distinct root causes of flakiness [12]. Studies on flakiness in Python or Rust provided additional evidence that some root causes of flakiness may be specific to certain programming languages [13, 37]. For example, categories such as Uninitialized variables can be particularly relevant for languages with lower-level memory management, such as C++ [12].

Description: `<test_suite_j>` fails FAIL: `<test_x>`
Root Cause: Ordered results expected, but unordered results produced.
Symptom: `<test_x>` fails sporadically.
Resolution: Expect unordered results.

Figure 1: An example issue report.

3 Study Context

SAP HANA is a large-scale in-memory database management system that has been developed for more than 10 years, consisting of millions of lines of code [2]. The SAP HANA repository is maintained by more than 100 active developers. [2]. Since SAP HANA commonly serves as the data management backbone of SAP's enterprise software [24], potential software failures can incur high costs [16]. To prevent such failures, SAP HANA is continuously and extensively tested.

SAP HANA's test suite consists of two types of tests: *native unit tests*, written in C++ , and Python *system tests* that communicate with a running database instance via SQL. Typically, unit tests have "low" execution times of a few seconds, whereas system tests exhibit "medium to large" execution times up to multiple hours [3]. Developers consider system tests to be more heavily affected by flakiness due to their larger scope and lower degree of isolation [6, 29]. Likewise, in SAP HANA's CI system, test failures trigger three repeated executions of the failing test to prevent breakages caused by flaky failures. Thus, flaky failures of system tests are also considered more costly, as they lead to higher wait times for developers and require more computational resources due to their increased execution times. In contrast to the typical testing pyramid, where a higher number of unit tests builds the foundation of a test suite, SAP HANA's developers focus more on system tests, as they provide valuable information on the system under test. Thus, both types of test play an important role in testing SAP HANA.

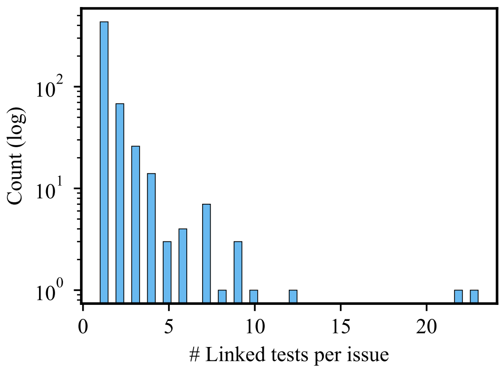

Figure 2: Histogram showing the number of linked tests per issue report. Note the logarithmic scale of the y-axis.

4 Dataset

We selected a dataset of 559 issue reports of fixed flaky tests sampled over a 34-month period between 2023 and 2025. These issue reports were tagged as related to flakiness and marked "resolved fixed" in the issue tracker. Each of the issue reports was linked to one or more tests. Figure 2 shows the distribution of test count per report. The number of tests per issue report ranges from 1 to 23. The issue reports were most commonly related to a single test (434 of 559 issue reports, 78%). Since some tests appear in multiple reports, our study covers 587 unique tests, each labeled either as a *system test* or a *native unit test*. We assigned each issue report to the respective test type (*system test*, *native unit test*, or *mixed*), yielding 439 issues related to system tests, 108 to native unit tests, and 12 mixed.

We utilized the following issue fields for this study: *Description*, *Root cause*, *Symptom*, and *Resolution* as shown in Figure 1. The fields for root cause and description are filled out when an issue report is filed and were filled out in all issue reports. In contrast, symptoms and resolutions are entered after an issue has been fixed. The symptom of the issue was available in 95% of the reports and the resolution was available in 73% of the reports.

5 Research Questions and Methodology

In this section, we introduce the research questions and the methodology we used to answer them.

RQ1 What are the prevalent root causes of flaky tests in SAP HANA issue reports for unit vs. system tests?

RQ2 Can we identify trends or patterns in the root causes of flaky tests in SAP HANA issue reports?

- **RQ2.1** How do root causes of flaky tests differ in issue reports linked to multiple tests?
- **RQ2.2** How do root causes of flaky tests change over time?

5.1 RQ1: Root Causes

5.1.1 Motivation. As described in Section 3, previous research indicates that the most common root causes of flakiness can vary depending on the type of test [12]. Furthermore, most of the existing approaches to automatically fix flaky tests are customized to fix specific root causes of flakiness [8, 21, 35, 41]; there seems to be no one-size-fits-all solution readily available. Therefore, before evaluating potential approaches to fix flakiness in the context of SAP HANA, our objective was to gain an overview of the root causes of flakiness. As described in Section 3, both system and native unit tests play an important role in testing SAP HANA. With the first research question, our aim is to determine whether there are differences in the root causes of flakiness between the two test types.

5.1.2 Approach. To gain an overview of flakiness root causes, we analyzed issue reports in SAP HANA's ticket system that are related to flaky tests. Similar to previous work [37], we focused our analysis on issue reports where the root cause of flakiness is already known, i.e., issue reports with the status "resolved fixed". To analyze issue reports, our approach combined automated and manual methods. First, we have manually annotated a subset of 232 tests to establish a ground truth for estimating the success rate of our automated approach. This sample size represents the minimum required to ensure that the resulting estimate lies within a 95% confidence interval (5% margin of error). Second, given an issue report, we automatically labeled the affected test(s) linked to the report as system or native unit tests based on the content of the respective test file. Comparing with the ground truth that we manually annotated, we obtained an accuracy of 96%, which we considered sufficient to proceed with our analysis. Finally, our automated approach clustered the 587 tests into 464 system tests and 123 C++ tests.

In addition to labeling tests as system tests or native unit tests, we combined a manual and an automated approach to categorize the root causes of the issue reports. We manually annotated a subset of the issue reports in our dataset. For automation, we used an *LLMs-as-annotators* approach as shown in Figure 3. Given the description, root cause, symptom, and resolution of an issue report, we repeatedly asked three different LLMs five times to "label the report into one of the following categories" using the categories described in Section 3. We considered a response from an LLM *valid* if it assigns the same category 4 out of 5 times. To obtain the final label, we performed a majority vote between the valid answers of the three LLMs (`gemini-2.5-pro`, `gpt-5`, `claude-4-sonnet`). We used a temperature of 0 for all three models to decrease the non-determinism of the responses [4, 27]. In an initial examination of the resulting LLM-generated labels, we found some cases where two root cause categories overlap. For example, some tests may only experience timeout issues on a specific platform. To conclude such cases, we added a code book to the prompt, which specifies the following cases:

(1) Prefer Platform if the error occurs only in a certain setup.
(2) Prefer Async Wait over Timeout if the timeout occurs during a method call within the test.
(3) Prefer Concurrency over Memory Management if memory-related issues arise due to, e.g., race conditions.

Finally, we quantified model-human and model-model inter-rater agreements. We first compared the labels of our three models using Fleiss' Kappa, a measure of agreement between multiple reviewers. Thus, we sought to understand the model-model agreement, which has been shown to correlate positively with human-model agreement in prior work [1]. For model-model agreement, we obtained $\kappa = 0.78$, which represents very good agreement [38]. All three models exhibit high internal consistency, i.e., provided the same answers in 4 out of 5 repeated requests in more than 88% of the cases. This is expected since we set the temperature to 0 to achieve the best possible determinism in the model responses. For the majority vote, we observed at least one valid answer for all issue reports, three valid answers for 438 of 559 reports (78%), and two valid answers for 545 of 559 reports (97%). In cases where only two models provided valid answers and the two answers disagreed (55), we resolved the disagreement through human judgment. To quantify the model-human agreement in our labeling, we used Cohen's Kappa to compare the majority voting results with human annotations. We achieved $\kappa = 0.63$, which represents substantial agreement [20]. Disagreements between the model and the human labels were mainly due to issue reports that could be assigned to multiple categories.

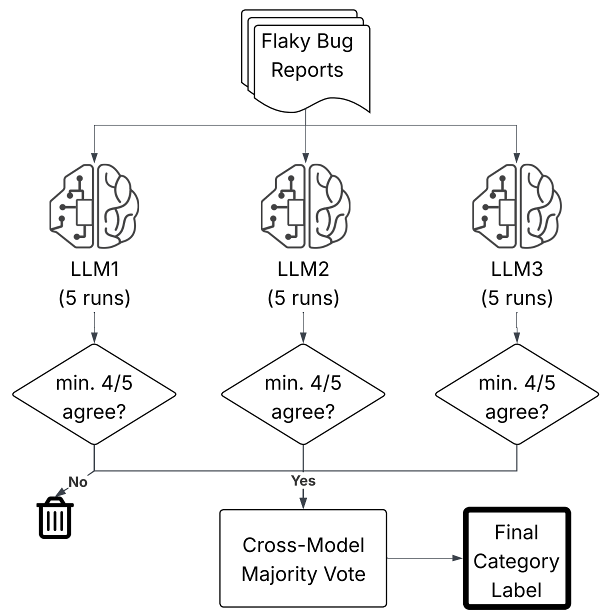

Figure 3: Our bug report labeling approach.

5.2 RQ2: Trends and Patterns

5.2.1 Motivation. Parry et al. [30] coined the term *systemic flakiness* to describe the co-occurrence of flaky failures, i.e., multiple tests failing flakily at the same time for the same reason. They identified network issues as a prevalent cause of such systemic flakiness and pointed out that awareness of co-occurring flakiness can reduce the cost of fixing flakiness by targeting multiple flaky tests with a single fix. With our second research question, our aim is to determine whether the root causes of flakiness differ when multiple tests are affected. In addition, we want to gain an overview of how the root causes of flaky tests change over time.

5.2.2 Approach. To investigate these two questions in SAP HANA issue reports, we analyzed the labeled data from RQ1. In contrast to Parry et al. [29], who clustered individual tests given their likelihood of co-occurring failures, we focus on identifying systemic flakiness on a root cause category level to identify whether certain root cause categories are more likely to produce co-occurring failures. We performed two different analyses. First, we investigated the root causes of issue reports with multiple linked tests (127) to determine whether there are root cause categories with a higher chance of affecting multiple tests. Second, we aggregated the number of issue reports for each root cause category into quarterly intervals to gain insight into trends and variation in the number of issue reports across root cause categories over time.

Table 2: Flakiness root causes per test type in our sample of issue reports ($n = 559$).

Root Cause	System	Unit	Mixed	Total
Concurrency	85 (19%)	42 (39%)	3 (25%)	130 (23%)
Timeout	77 (18%)	8 (7%)	2 (17%)	87 (16%)
Oracle-Brittleness	49 (11%)	7 (6%)	1 (8%)	57 (10%)
Configuration	49 (11%)	6 (5%)	1 (8%)	55 (10%)
Async Wait	42 (10%)	2 (2%)	1 (8%)	52 (9%)
Isolation	28 (6%)	13 (12%)	2 (17%)	43 (8%)
Platform	25 (4%)	13 (12%)	1 (8%)	39 (7%)
Application	35 (8%)	1 (1%)	0 (0%)	36 (6%)
Test-Framework	12 (3%)	5 (5%)	0 (0%)	17 (3%)
Error-Handling	14 (3%)	1 (1%)	1 (8%)	16 (3%)
Memory-Management	9 (2%)	4 (4%)	0 (0%)	13 (2%)
Environment	9 (2%)	2 (2%)	0 (0%)	11 (2%)
Randomization	1 (0%)	3 (3%)	0 (0%)	4 (1%)
Network	3 (1%)	0 (0%)	0 (0%)	3 (0%)
Unknown	0 (0%)	2 (2%)	0 (0%)	2 (0%)
Numerical-Imprecision	1 (0%)	0 (0%)	0 (0%)	1 (0%)
Locale	1 (0%)	0 (0%)	0 (0%)	1 (0%)
Total	439 (100%)	108 (100%)	12 (100%)	559 (100%)

6 Results

In this section, we present our results.

6.1 RQ1: Root Causes

As shown in Table 2, the most common root cause in our dataset is concurrency, which appears in 130 of 559 issue reports (23%). This finding is in line with existing research that identified concurrency as a major cause of test flakiness [18, 22, 39]. Comparing test types, we observe that more than one out of three issue reports for native unit tests is related to concurrency (39%). However, for system tests, this proportion is considerably lower (19%). Instead, system tests appear to suffer more from timeout-related issues (18% of issue reports) than native unit tests (7%). This can be explained by an architectural difference in SAP HANA's test framework. The execution of native unit tests is globally canceled after one hour, with none of the native unit tests usually running for close to one hour.

In contrast, the system tests were formerly assigned a dedicated per-test-timeout by developers, which was close to the test's average execution time. This architectural difference was harmonized at the beginning of 2024 [6], which represents approximately half of the time interval in our study. Looking at the data after harmonization, we find that the proportion of timeout-related issues for system and native unit tests is similar.

Furthermore, while Oracle Brittleness and Configuration appear as common categories for system tests, native unit tests tend to suffer more from flakiness caused by Isolation and Platform. These findings seem intuitive. Since system tests require a running database, their result may be influenced by sophisticated configuration settings of this database instance. Based on an analysis of issue reports related to Oracle Brittleness of system tests, we observe that missing `ORDER BY` statements or brittle exact string comparisons are common problems.

In contrast, native unit tests are commonly used for testing low-level functionality, which can depend on specific features of the executing platform such as Non Uniform Memory Access (NUMA) support, thus explaining the high number of *Platform* flakiness.

Answer RQ1 (Root causes): Concurrency is the most common root cause of flakiness for both test types in the given dataset, appearing in 130 of 559 cases (23%). Regarding differences between test types, system tests suffer from Timeouts in 18% of the reports and Oracle Brittleness in 11% of issue reports. In contrast, native unit tests suffer more commonly from Platform (12%) and Isolation (12%).

6.2 RQ2: Trends and Patterns

With our second research question, we aim to quantify trends and patterns of root cause categories for SAP HANA's flaky tests. To achieve this, we first analyzed 127 issue reports related to more than one test. Of these 127 issue reports, 94 were related to Python tests, 21 to native unit tests, and the remaining 12 depict mixed reports linked with both Python and native unit tests.

Figure 4 shows the number of issue reports per root cause category for reports with multiple tests. Figure 4 shows that Concurrency remains the most common root cause category for both test types, being present in 45 of 127 reports (35%). Relative to issue reports concerning only one test, we observe frequent occurrences of *Platform* flakiness. In fact, the proportion of reports related to Platform rises from 7% to 13%. In contrast, the proportion of *Oracle Brittleness* drops from 10% to 6%. The relevance of issues related to Configuration remains at 10%.

Figure 5 displays the number of issue reports, aggregated per category per quarter of the year. To emphasize the variations within issue reports of a category over time, we min-max-normalized the issue report counts within each category. Overall, we observe that common root cause categories such as Concurrency or Oracle Brittleness exhibit comparatively constant rates of related issue reports throughout the examined time interval. For example, while the number of issue reports related to Concurrency ranges between 3 and 16 per quarter, we observe more than 10 reports per

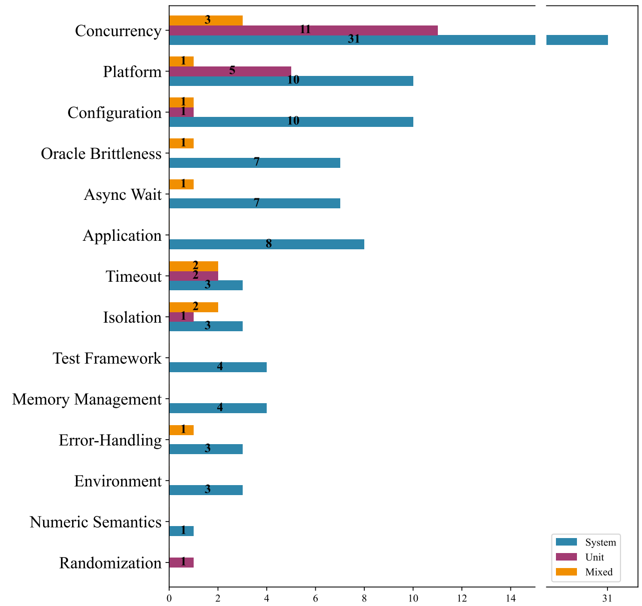

Figure 4: Barplot showing the number of issue reports per category per test type. Includes only issue reports with more than a single linked test (n=127).

quarter in 9 out of 11 quarters. The pattern for Oracle Brittleness appears similar, but on a lower scale. Ranging between 1 and 8 reports per quarter throughout the examined time interval, there were 5 or more reports per quarter in 7 of 11 quarters. Examining the number of issue reports related to Timeout, we observe a sharp drop after the architectural change towards a global timeout value, which confirms our expectations described in Section 6.1. Aside from Timeout, the most noticeable leap can be observed for Async Wait. Although typically ranging between 1 and 6 reports per quarter, this number rises by a factor of 2.2 for 2024-Q2 (13 issue reports).

Answer RQ2 (Trends): For issue reports related to multiple tests, Concurrency remains the most common root cause category in the given dataset. Relative to issue reports concerning only one test, the proportion of reports related to Platform flakiness rose from 7% to 12%. Over time, the most noticeable rise in issue reports in a quarter was for Async wait, which rose by a factor of 2.2 compared to the other quarters. Concurrency and Oracle Brittleness showed a rather constant rate of reported issues. The rate of Timeout related issues dropped notably after an intervention.

7 Discussion

On LLMs as annotators. In accordance with previous work on the use of LLMs as annotators [1], we found both the resulting inter-model reliability, the human-model reliability, and the intra-model consistency helpful for signaling the reliability of the resulting labels. Overall, we consider the efficiency gains from LLM-generated

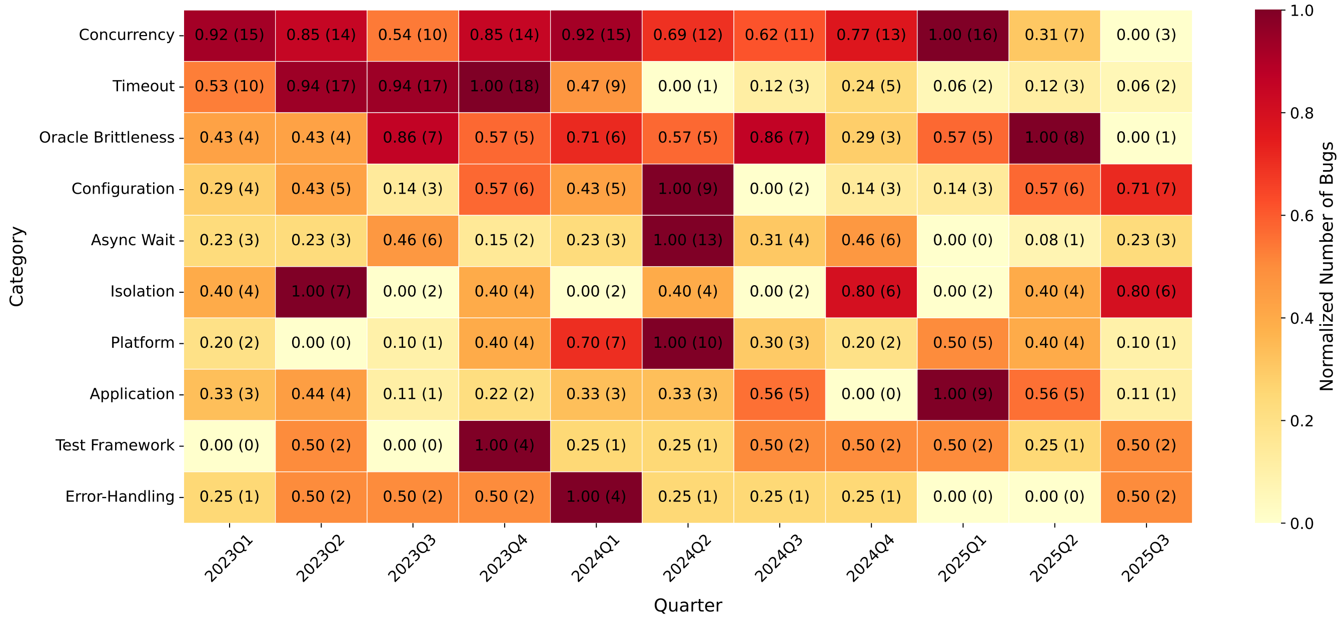

Figure 5: Heatmap showing the number of issue reports per category per quarter. Values are min-max-normalized within each category; absolute values are shown in parentheses. Note that some categories (rows) show high numbers throughout the period while others show more concentrated failure numbers.

labels helpful, as they enable a broad understanding of relevant root causes for flakiness at scale with reasonable effort.

On root cause labeling. As mentioned in Section 5, we added a code book to align LLM labels with human labels, increasing the reliability of the human-model from 0.50 to 0.63. As mentioned in Section 5, the instructions in our code book addressed mainly cases where flaky failures were caused by multiple contributing factors rather than an isolated cause that could be pinpointed. For example, a test might only face timeout issues on a specific platform, or race conditions might only occur when certain environmental conditions are met. Such co-occurrences of different influencing factors can cause complex reproductions of flaky failures, which have been reported as a major problem of flakiness in previous work [9, 19]. This overlap of different categories of flakiness root causes has also complicated prior work on automated flakiness categorization [34]. Future work could examine the idea that root-causing flaky tests is a multi-label problem, in which flaky failures result from a combination of contributing factors.

On the resulting root causes. Based on the labels given by the LLMs, we identified *Concurrency* as the most common root cause in the given dataset. This result seems intuitive, since compared to other software projects, SAP HANA is highly dependent on parallelization within the product code as well as for testing. Regarding the generalizability of our results, it is important to note that the ticket system from which our issue reports are drawn is primarily used by developers. As a result, global issues related to the test infrastructure are likely underrepresented in our dataset, as problems that developers can address are typically reported.

8 Conclusions

In this study, we empirically analyzed issue reports related to flakiness in the context of SAP HANA. To gain an overview of prevalent root causes of flakiness in that context, we explored an *LLMs as annotators* approach to divide issue reports into different root cause categories derived from previous work. Based on a comparison with a manually labeled sample, our results suggest that LLMs exhibit reasonable human-model alignment when automatically categorizing issue reports related to flakiness.

As previous research has suggested that the root causes of flakiness are context-dependent, we analyzed the root causes for unit and system tests separately. Our analysis reveals that both test types are primarily affected by Concurrency issues, including race conditions and improper thread handling. Regarding the differences between the tests, our results indicate that SAP HANA's system tests tend to experience flaky failures due to Oracle Brittleness, caused by improper use of `ORDER BY` or brittle exact string matching. In contrast, unit tests are more susceptible to Platform-specific details, such as compiler differences.

We encourage future work to elaborate on the idea of flakiness root causes as a multi-label problem. Instead of pinpointing flakiness down to a single root cause, it is valuable to gain more insights into interdependencies that cause flakiness. We believe that, especially in the context of large-scale software projects, identifying common problems caused by such interdependencies can help build future tooling to detect and fix flaky tests or flaky test executions.

References

[1] Toufique Ahmed, Premkumar T. Devanbu, Christoph Treude, and Michael Pradel. 2025. Can LLMs Replace Manual Annotation of Software Engineering Artifacts?. In *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 526–538. doi:10.1109/MSR66628.2025.00086

[2] Thomas Bach, Artur Andrzejak, Changyun Seo, Christian Bierstedt, Christian Lemke, Daniel Ritter, Dongwon Hwang, Erda Sheshi, Felix Schabernack, Frank Renkes, Gordon Gaumnitz, Jakob Martens, Lars Hömke, Michael Felderer, Michael Rudolf, Neetha Jambigi, Norman May, Robin Joy, Ruben Scheja, Sascha Schwedes, Sebastian Seibel, Sebastian Seifert, Stefan Haas, Stephan Kraft, Thomas Kroll, Tobias Scheuer, and Wolfgang Lehner. 2022. Testing Very Large Database Management Systems: The Case of SAP HANA. *Datenbank-Spektrum* 22, 3 (2022), 195–215. doi:10.1007/S13222-022-00426-X

[3] Thomas Bach, Ralf Pannemans, Johannes Häussler, and Artur Andrzejak. 2019. Dynamic Unit Test Extraction via Time Travel Debugging for Test Cost reduction. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 238–239. doi:10.1109/ICSE-COMPANION.2019.00093

[4] Sebastian Baltes, Florian Angermeir, Chetan Arora, Marvin Muñoz Barón, Chunyang Chen, Lukas Böhme, Fabio Calefato, Neil Ernst, Davide Falessi, Brian Fitzgerald, Davide Fucci, Marcos Kalinowski, Stefano Lambiase, Daniel Russo, Mircea Lungu, Lutz Prechelt, Paul Ralph, Rijnard van Tonder, Christoph Treude, and Stefan Wagner. 2025. Guidelines for Empirical Studies in Software Engineering involving Large Language Models. arXiv:2508.15503 [cs.SE] https://arxiv.org/abs/2508.15503

[5] Alexander Berndt, Thomas Bach, and Sebastian Baltes. 2024. Do Test and Environmental Complexity Increase Flakiness? An Empirical Study of SAP HANA. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2024, Barcelona, Spain, October 24-25, 2024*, Xavier Franch, Maya Daneva, Silverio Martínez-Fernández, and Luigi Quaranta (Eds.). ACM, 572–581. doi:10.1145/3674805.3695407

[6] Alexander Berndt, Sebastian Baltes, and Thomas Bach. 2024. Taming Timeout Flakiness: An Empirical Study of SAP HANA. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 69–80. doi:10.1145/3639477.3639741

[7] Alexander Berndt, Zoltán Nochta, and Thomas Bach. 2023. The Vocabulary of Flaky Tests in the Context of SAP HANA. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2023, New Orleans, LA, USA, October 26-27, 2023*. IEEE, 1–9. doi:10.1109/ESEM56168.2023.10304860

[8] Yang Chen and Reyhaneh Jabbarvand. 2024. Neurosymbolic Repair of Test Flakiness. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 1402–1414. doi:10.1145/3650212.3680369

[9] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. 2019. Understanding Flaky Tests: The Developer's Perspective. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 830–840. doi:10.1145/3338906.3338945

[10] Emad Fallahzadeh and Peter C. Rigby. 2022. The Impact of Flaky Tests on Historical Test Prioritization on Chrome. In *44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2022, Pittsburgh, PA, USA, May 22-24, 2022*. IEEE, 273–282. doi:10.1109/ICSE-SEIP55303.2022.9793941

[11] Sakina Fatima, Hadi Hemmati, and Lionel C. Briand. 2024. FlakyFix: Using Large Language Models for Predicting Flaky Test Fix Categories and Test Code Repair. *IEEE Trans. Software Eng.* 50, 12 (2024), 3146–3171. doi:10.1109/TSE.2024.3472476

[12] Martin Gruber and Gordon Fraser. 2022. A Survey on How Test Flakiness Affects Developers and What Support They Need To Address It. In *15th IEEE Conference on Software Testing, Verification and Validation, ICST 2022, Valencia, Spain, April 4-14, 2022*. IEEE, 82–92. doi:10.1109/ICST53961.2022.00020

[13] Martin Gruber, Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. 2021. An Empirical Study of Flaky Tests in Python. In *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12-16, 2021*. IEEE, 148–158. doi:10.1109/ICST49551.2021.00026

[14] Guillaume Haben, Sarra Habchi, John Micco, Mark Harman, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2024. The Importance of Accounting for Execution Failures when Predicting Test Flakiness. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, 1979–1989. doi:10.1145/3691620.3695261

[15] Mark Harman and Peter W. O'Hearn. 2018. From Start-ups to Scale-ups: Opportunities and Open Problems for Static and Dynamic Program Analysis. In *18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018, Madrid, Spain, September 23-24, 2018*. IEEE Computer Society, 1–23. doi:10.1109/SCAM.2018.00009

[16] Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The Art of Testing Less without Sacrificing Quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 483–493. doi:10.1109/ICSE.2015.66

[17] Minh Hoang and Adrian Berding. 2024. Presubmit Rescue: Automatically Ignoring FlakyTest Executions. In *Proceedings of the 1st International Workshop on Flaky Tests, FTW 2024, Lisbon, Portugal, 14 April 2024*. ACM, 1–2. doi:10.1145/3643656.3643896

[18] Wing Lam, Patrice Godefroid, Suman Nath, Anirudh Santhiar, and Suresh Thummalapenta. 2019. Root Causing Flaky Tests in a Large-scale Industrial Setting. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, Dongmei Zhang and Anders Møller (Eds.). ACM, 101–111. doi:10.1145/3293882.3330570

[19] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, and Darko Marinov. 2020. Understanding Reproducibility and Characteristics of Flaky Tests Through Test Reruns in Java Projects. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng (Eds.). IEEE, 403–413. doi:10.1109/ISSRE5003.2020.00045

[20] J Richard Landis and Gary G Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *biometrics* (1977), 159–174.

[21] Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, and Weihang Wang. 2024. WEFix: Intelligent Automatic Generation of Explicit Waits for Efficient Web End-to-End Flaky Tests. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 3043–3052. doi:10.1145/3589334.3645628

[22] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An Empirical Analysis of Flaky Tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey (Eds.). ACM, 643–653. doi:10.1145/2635868.2635920

[23] Benjamin Magill and Phil McMinn. 2025. deflake.rs: Detect Flaky Tests in Rust Projects using Execution Data. *J. Open Source Softw.* 10, 113 (2025), 8757. doi:10.21105/JOSS.08757

[24] Norman May, Alexander Böhm, Daniel Ritter, Frank Renkes, Mihnea Andrei, and Wolfgang Lehner. 2025. SAP HANA Cloud: Data Management for Modern Enterprise Applications. In *Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025, Berlin, Germany, June 22-27, 2025*, Volker Markl, Joseph M. Hellerstein, and Azza Abouzied (Eds.). ACM, 580–592. doi:10.1145/3722212.3724452

[25] Atif M. Memon, Zebao Gao, Bao N. Nguyen, Sanjeev Dhanda, Eric Nickell, Rob Siemborski, and John Micco. 2017. Taming Google-Scale Continuous Testing. In *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 233–242. doi:10.1109/ICSE-SEIP.2017.16

[26] Olek Osikowicz, Phil McMinn, and Donghwan Shin. 2025. Empirically Evaluating Flaky Tests for Autonomous Driving Systems in Simulated Environments. In *IEEE/ACM International Flaky Tests Workshop, FTW@ICSE 2025, Ottawa, ON, Canada, April 27, 2025*. IEEE, 13–20. doi:10.1109/FTW66604.2025.00009

[27] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2025. An Empirical Study of the Non-Determinism of ChatGPT in Code Generation. *ACM Trans. Softw. Eng. Methodol.* 34, 2 (2025), 42:1–42:28. doi:10.1145/3697010

[28] Owain Parry, Michael Hilton, Gregory M. Kapfhammer, and Phil McMinn. 2022. What Do Developer-Repaired Flaky Tests Tell Us About the Effectiveness of Automated Flaky Test Detection?. In *IEEE/ACM International Conference on Automation of Software Test, AST@ICSE 2022, Pittsburgh, PA, USA, May 21-22, 2022*. ACM/IEEE, 160–164. doi:10.1145/3524481.3527227

[29] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. 2022. A Survey of Flaky Tests. *ACM Trans. Softw. Eng. Methodol.* 31, 1 (2022), 17:1–17:74. doi:10.1145/3476105

[30] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. 2025. Systemic Flakiness: An Empirical Analysis of Co-Occurring Flaky Test Failures. *CoRR* abs/2504.16777 (2025). arXiv:2504.16777 doi:10.48550/ARXIV.2504.16777

[31] Kai Presler-Marshall, Eric Horton, Sarah Heckman, and Kathryn T. Stolee. 2019. Wait wait. No, tell me: Analyzing Selenium Configuration Effects on Test Flakiness. In *Proceedings of the 14th International Workshop on Automation of Software Test, AST@ICSE 2019, May 27, 2019, Montreal, QC, Canada*, Byoungju Choi, María José Escalona, and Kim Herzig (Eds.). IEEE / ACM, 7–13. doi:10.1109/AST.2019.000-1

[32] The Chromium Project. 2020. Test Flakiness - One of the Main Challenges of Automated Testing. https://testing.googleblog.com/2020/12/test-flakiness-one-of-main-challenges.html Accessed 2025-10-18.

[33] The Chromium Project. 2025. Fixing Flaky Unit Tests - Developer Guidelines. https://www.chromium.org/developers/testing/fixing-flaky-tests/fixing_flaky_unittests/ Accessed 2025-10-18.

[34] Shanto Rahman, Saikat Dutta, and August Shi. 2025. Understanding and Improving Flaky Test Classification. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 320 (Oct. 2025), 27 pages. doi:10.1145/3763098

[35] Shanto Rahman and August Shi. 2024. FlakeSync: Automatically Repairing Async Flaky Tests. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 136:1–136:12. doi:10.1145/3597503.3639115

[36] Alan Romano, Zihe Song, Sampath Grandhi, Wei Yang, and Weihang Wang. 2021. An Empirical Analysis of UI-based Flaky Tests. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1585–1597. doi:10.1109/ICSE43902.2021.00141

[37] Tom Schroeder, Minh Phan, and Yang Chen. 2025. A Preliminary Study of Fixed Flaky Tests in Rust Projects on GitHub. In *IEEE/ACM International Flaky Tests Workshop, FTW@ICSE 2025, Ottawa, ON, Canada, April 27, 2025*. IEEE, 21–22. doi:10.1109/FTW66604.2025.00010

[38] Julius Sim and Chris C Wright. 2005. The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements. *Physical therapy* 85, 3 (2005), 257–268.

[39] Swapna Thorve, Chandani Sreshtha, and Na Meng. 2018. An Empirical Study of Flaky Tests in Android Apps. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 534–538. doi:10.1109/ICSME.2018.00062

[40] Arash Vahabzadeh, Amin Milani Fard, and Ali Mesbah. 2015. An Empirical Study of Bugs in Test Code. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 101–110. doi:10.1109/ICSM.2015.7332456

[41] Peilun Zhang, Yanjie Jiang, Anjiang Wei, Victoria Stodden, Darko Marinov, and August Shi. 2021. Domain-Specific Fixes for Flaky Tests with Wrong Assumptions on Underdetermined Specifications. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 50–61. doi:10.1109/ICSE43902.2021.00018