What we talk about when we talk about software test flakiness*

Morena Barboni$^{1[0000-0002-1281-4058]}$✉, Antonia Bertolino$^{2[0000-0001-8749-1356]}$, and Guglielmo De Angelis$^{1[0000-0002-1076-0076]}$

[1] IASI-CNR, Rome, Italy
{morena.barboni, guglielmo.deangelis}@iasi.cnr.it
[2] ISTI-CNR, Pisa, Italy
antonia.bertolino@isti.cnr.it

Abstract. Software test flakiness is drawing increasing interest among both academic researchers and practitioners. In this work we report our findings from a scoping review of white and grey literature, highlighting variations across flaky tests key concepts. Our study clearly indicates the need of a unifying definition as well as of a more comprehensive analysis for establishing a conceptual map that can better guide future research.

Keywords: Flaky Tests · Flakiness · Software Testing · Scoping Review

1 Introduction

In recent years research aiming at understanding and mitigating the problem of test flakiness has boomed, also pushed by alarms raised by big companies as Google [29], Facebook [27] or Apple [17], among others, on the extent and cost of this phenomenon.

However, in the fast rising of articles addressing the theme, researchers provide different characterizations and modelings of the involved aspects and connected techniques. Terms such as *flaky* or *intermittent* or *non-deterministic* are used by some as synonyms, by others to identify differing test behaviors. Some works study in depth the causes of flakiness and introduce more specific test characterizations. However, as it is inevitable when many authors work in concurrence, a same concept is introduced in more articles using differing terms.

Lack of a shared terminology and of an agreed conceptual scheme may be confounding and may also waste effort in re-inventing existing knowledge. For instance, already in the 90's Carver and Tai [6] warned that multiple executions of a concurrent program under a same test input might produce different results, if the underlying sequence of synchronization events is not specified. This sounds to us closely related with those flaky test categories commonly classified as due to concurrency or asynchronous wait [26], but to the best of our knowledge no recent article has ever acknowledged the evident connection. On the other hand,

* Work supported by the Italian MIUR PRIN 2017 Project: SISMA (Contract 201752ENYB), and partially by the Italian Research Group: INdAM-GNCS.

our analysis of the literature also emphasizes how the same term of a "flaky test" can refer to situations that require different treatments, e.g., in some cases fixing the test code, in other ones refining the test environment configuration. For all such reasons, we think that putting order in the fuzz around test flakiness can be helpful to better guide future research efforts.

In this short paper we highlight the problem of inconsistent terminology based on a scoping review of literature [3, 30] (Section 2), and move some first steps towards proposing a unifying definition and vocabulary for test flakiness (Section 3), which will be the aim of our future work (Section 4).

2 Scoping Review of White and Grey Literature

This study aims to examine definitions and key concepts related to software testing flakiness. For this purpose we borrow from medical research the recently introduced approach of a *scoping review* [3, 30]. Similarly to systematic review, which is a better known methodology in software engineering [16], a scoping review must apply a defined and repeatable search protocol. However, scoping studies[3] do not address the lengthy synthesis stage, aiming rather at a fast descriptive appraisal of broad questions, often as a precursor to deeper systematic reviews. As our goal was to highlight variations behind definitions of flaky tests, our search methodology relied on two pragmatic criteria: *i)* it should cover both white and grey literature, as the phenomenon of flakiness has raised great interest from both academic researchers and practitioners; and *ii)* for the sake of timeliness, we established well-delimited boundaries to our search (explained below). While these limitations may hinder comprehensiveness, nevertheless our results were already sufficient to find inconsistencies, as we discuss later in this section.

Search Methodology: The entries from the white literature have been retrieved by consulting three among the most relevant Academic Digital Libraries in Software Engineering, namely: Scopus, ACM Digital Library, IEEE eXplore. For the analysis of the grey literature, we rely on articles posted on `Medium.com`, a well-known online publishing platform hosting many informal technology blogs that are frequently written/followed by Software companies and practitioners.

We launched an automated search on the white literature sources by querying: title, abstract and keywords; matching with: `"Flaky test" OR "Flakiness"`, and selecting those *English papers* published until Feb. 2021 (date of the query). From the collected entries, it appears evident that before Luo et al.'s paper [26], there were only few works explicitly referring to software testing "flakiness".

Then, the same query string has been run on Google Search but limiting the search space to `Medium.com` and by considering the top-ten returned *English articles* written until Apr. 2021 (date of the query).

Even though we did not include a formal snowballing process, during the analysis of the collected entries, two blog posts, namely [12] and [29], stood out

[3] A useful comparison between systematic reviews, scoping reviews and other review types is available from `https://guides.temple.edu/systematicreviews`.

as playing a seminal role, and therefore we decided to also include them among the grey literature entries.
Overview of Findings: Interestingly, our preliminary analysis of the collected entries revealed quite different implications and perspective on how literature conceives flaky tests. Indeed there are works that explicitly refer to their randomised nature: "*Flaky tests are software tests that exhibit a seemingly random outcome (pass or fail) despite exercising unchanged code*" [10], but also others that definitively reject such vision: "*They are sometimes referred to as random failures, but in reality, it's often less about actual randomness than very reproducible edge cases that happen in a seemingly random fashion*" [41].

In addition, along with definitions that exclude any evolution exclusively referring to the software under test (SUT) as in [10] above, others also cover testing configurations or the execution environment: "*A flaky test is a test that can be failing or passing with no changes in the application or infrastructure*" [40].

Finally, most works depict flakiness as an undesired behavior of test programs, but few others highlight how it may lead developers to disclose potential bugs not revealed otherwise: "*Part of the test or production code has a non-deterministic outcome*" [31].

For lack of space, the complete list of the collected definitions is made available online [4].

3 Commonalities in Test Flakiness Concepts

Based on the outcome of our scoping review, in the following subsections we aim to bring some order to flaky test-related concepts (Section 3.1) and classifications (Section 3.2), taking a first step towards the identification of a more consistent vocabulary.

3.1 Definitions of Flaky Test Concepts

Table 1 shows a synthesis of the terminology actively used by both academics and practitioners. This summary was derived after a careful sampling and analysis of concepts related to the behavior of test outcomes. The most recurring definition of **Flaky Test** from the literature is (1) *"a test that exhibits a non-deterministic behavior"*. This suggests that any test showing both pass and failure outcomes upon multiple repeated executions is usually marked as flaky. Conversely, **Not Flaky** tests are defined by Lam et al. [22] as tests that either always pass or always fail in a deterministic manner, whereby tests that always exhibit a failure outcome can be further classified as **Consistently Failing Tests** [42]. The most common synonym for flaky test is **Non-Deterministic Test** [12]. However, we observed that considering "*flaky*" and "*non-deterministic*" as interchangeable terms can be a source of confusion. In fact, as we discuss in Section 3.2, the term Non-Deterministic is also used for designating a very specific subclass of flaky tests. On several occasions, flaky tests are also referred to as (3) "*tests that fail intermittently*". Although the manifestation of intermittent outcomes

Table 1. Definitions of Flaky Test Concepts

Term	**Definition**	**Source(s)**
Non-Flaky	A test that either always passes or always fails.	[22]
Flaky	(1) A test that exhibits a non-deterministic behavior.	[17, 25, 27, 31, 34, 37, 39, 45]
	(2) A test that provides different results inconsistently.	[41]
	(3) A test that fails intermittently.	[10, 26, 46]
	(4) A test that fails randomly.	[7]
	(5) A test that exhibits pass and failure outcomes despite exercising *unchanged code.*	[2, 5, 9, 10, 19, 24, 26, 29, 32, 33, 35, 36, 44]
	(6) A test that exhibits pass and failure outcomes although *neither the code nor the test has changed.*	[11, 15, 21, 49]
	(7) A *regression* test that exhibits pass and failure outcomes *although neither the code nor the test has changed.*	[13]
	(8) A test that exhibits pass and failure outcomes *although neither the code nor the test infrastructure has changed.*	[40]
	(9) A test that exhibits pass and failure outcomes although *neither the code nor the configuration has changed.*	[18]
	(10) A test that exhibits pass and failure outcomes although *the code, the inputs and the configuration have not changed.*	[43]
	(11) A test that exhibits pass and failure outcomes although *the SW, the HW and the TW have not changed.*	[42]‘
	(12) A test that exhibits pass and failure outcomes while exercising a *potentially changed version of the code.*	[20, 23, 26, 38]
	(13) A test that exhibits pass and failure outcomes in *apparently identical test scenarios.*	[28]
	(14) A test that exhibits pass and failure outcomes although *neither the test code nor the configuration parameters has changed.*	[48]
	(15) A test that exhibits pass and failure outcomes while exercising a *potentially changed version of the code* and a *potentially evolved test environment.*	[22]
Non-Deterministic	A test exhibits both pass and failure outcomes without any noticeable change in the code, tests, or environment.	[12]
Latent Flaky	A test that is not currently flaky, but that could become so due to a latent source(s) of flakiness.	[33]
Intermittently Failing	A test that exhibits pass and failure outcomes while there has been a potential evolution in the SW, the HW or the TW.	[42]
Consistently Failing	A test that exhibits a consistent failure outcome.	[42]

can accurately depict the behavior of flaky tests, Strandberg et al. [42] explicitly differentiate **Intermittently Failing Tests** from the former. We also observed flaky tests being described as (4) "*tests that fail randomly*". However, as specified by Stosik [41], this definition is imprecise because the randomness of flaky tests is only apparent. Indeed, the (2) "inconsistent behavior" of a flaky test is often caused by a well-defined, reproducible set of conditions. Even though commonly accepted, definitions (1-3) do not provide any insight relative to the context in which the test exhibits an inconsistent behavior. In particular, it is unclear whether the flakiness is associated to problems in the test code, in the SUT, or to any other environmental factor. Moreover, they do not specify explicitly whether any element, such as the SUT or the test code, underwent any type of modification across different test re-runs.

Several works extend these generic definitions with additional details. In particular, definitions (5 - 11) agree upon the fact that flaky tests *"exhibit both pass and failure outcomes despite exercising unchanged code"*. The fact that a given test can return non-deterministic outcomes for the same code version was identified as one of the main obstacles for regression testing activities. Whenever a developer updates the code, the tests are re-run to ensure that said changes do not break existing functionalities. A regression test failure normally indicates the (re)introduction of a bug that impacts previously working software. However, a test that non-deterministically passes and fails for the same code version provides misleading signals to the developer, who might waste considerable time and effort in debugging the code under test. While definition (5) only makes assumptions related to the SUT, definitions (6 - 11) provide further constraints as to what constitutes a flaky test. In particular, definitions (6) and (7) require both the SUT and the test code to be unchanged, although the latter explicitly identifies a flaky test as a type of regression test. Other authors require the test environment (8), the configuration (9, 10) and the inputs (10) to stay the same upon multiple test executions. The definition (11) proposed by Strandberg et al. [42] specifies that a flaky test yields differing verdicts when *"nothing in the SW, HW or TW has been changed."*. This vision of flakiness stems from the analysis of test intermittence in industrial Embedded Systems (ES), which comprise hardware (HW), software (SW), and testware (TW). The TW includes both software and hardware components, such as test libraries and the physical environment on which the tests are executed. This definition implies that flaky test verdicts are not caused by modifications to the aforementioned Embedded System components. Instead, flakiness can be due to "hidden" state or environment changes that might have occurred since the previous test run.

The vision of test flakiness described in definitions (12 - 15) is quite different, in that they do not require the immutability of the code under test. In particular, definition (12) explicitly admits changes to the code under test among multiple test re-runs. Definitions (13, 14) require the same test scenarios, and the same test and configuration respectively, but they do not explicitly ask for unchanged software. Lastly, definition (15) admits "*a potentially changed version of the code and a potentially evolved test environment*". We observed that the idea of test flakiness emerging from definitions (12 - 15) is partly reflected by Strandberg et al.'s [42] definition of Intermittently Failing Test for the Embedded Systems domain. As introduced earlier, the authors provide a separate definition for Intermittently Failing Tests, in that Flaky Tests as defined in (11) can be rarely observed in practice. Indeed, industrial ES undergo rapid and frequent changes during their development process. Conversely, an **Intermittently Failing Test** provides different verdicts over time, but it *"allows changes in the SW or HW of the ES under test, as well as in the TW used for testing"*. Definition (15) used in more traditional software systems is particularly in line with this vision of test intermittence, as it also admits changes in the code and test environment. To complete this preliminary categorization, we also report the concept of **Latent Flaky Test** proposed by Parry et al. [33]. A Flaky Test is said to be latent if it contains a source of flakiness that has not yet manifested. The concept of latent flakiness brings further attention to the problem of exposing test flakiness as soon as possible, so as to improve the reliability of the test suite.

3.2 Classification of Flaky Tests

As for different types of flakiness, many works broadly split flaky tests into two groups based on the underlying source of flakiness (Table 2). The dashed line denotes the separation of different classifications of flaky tests that we encountered during our research. **Order-Dependent (OD)** tests are usually described as tests that "*can pass or fail based on the order in which they are run*". The unreliability of OD test verdicts is generally caused by their reliance on some environment state that has been improperly (re)set by another test execution. Although the research community seems to agree on the concept of OD test, a minority of works [20, 22] further extend definition (1), specifying that OD tests actually exhibit deterministic behavior. In other words, an OD test either always passes or always fails for each order of tests, and there exist at least two orders for which it provides different verdicts. The flaky tests that do not match this requirement are commonly identified as **Non Order-Dependent (NOD)**. A typical example would be a test that uses the result of an asynchronous call without waiting for it to be ready. Based on the availability of the requested resource, the test can non-deterministically pass (or fail) regardless of its execution order. The possible root causes of flakiness for a NOD test are plentiful, but discussing them is out of scope for this work, as they have already been broadly investigated and analyzed in the literature [1, 10, 18, 26, 42, 44, 49]. During our research, we also encountered the term **Non-Deterministic (ND)** being used for identifying a test that "*passes or fails with no changes to test execution order*". Therefore, using "flaky" and "non-deterministic" interchangeably might be confusing in certain contexts, as OD tests can be flaky whilst providing a deterministic outcome for a specific order. Conversely, NOD tests always show a non-deterministic outcome regardless of the order in which they are run(1). Lam et al. [22] provide a more specific definition of NOD tests (2), hinting at an underlying order-dependency. Depending on the failure rate associated to each

Table 2. Classification of Flaky Tests

Order-Dependent Tests		
Term	**Definition**	**Source(s)**
OD	(1) A test that can pass or fail based on the order in which it is run.	[2, 10, 14, 21, 23, 26, 36, 38, 47]
	(2) A test that can *deterministically* pass or fail based on the order in which it is run.	[20, 22]
Victim	OD test that always passes when run in isolation from other tests.	[23, 38]
Brittle	OD test that consistently fails when run in isolation from other tests.	[23, 38]
Non Order-Dependent Tests		
Term	**Definition**	**Source(s)**
NOD	(1) A test that non-deterministically passes and fails regardless of its execution order.	[2, 20]
	(2) A test that non-deterministically passes and fails for at least one execution order.	[22]
ND	A test that non-deterministically passes or fails with no changes to test execution order or implementation of test dependencies. .	[23]
NDOD	NOD test where at least one order's failure rate significantly differs from other orders' failure rates.	[22]
NDOI	NOD tests where all failure rates do not significantly differ.	[22]
ID	A test whose outcome depends on the implementation of a non-deterministic specification.	[23]
Smelly	A test that might be flaky due to the presence of a test smell (i.e. a bad testing practice).	[1, 2, 42]

order, they classify a NOD test as either a NDOI or a NDOD, which we discuss later.

Classification of OD Tests According to several works [23, 38], there exist two different types (see Table 2) of OD flaky tests: Victim (OD-Vic) and Brittle (OD-Brit). The difference between the two lies in the behavior of the OD test when run in isolation from the test suite. If an OD test "*consistently passes when run by itself, but fails when run in combination with some other test(s)*", then it is marked as a **Victim**. Indeed, it suffers the consequences of executing other tests (i.e., polluters) that modify the test environment state without cleaning it up. On the other hand, a **Brittle** "*fails when run in isolation but passes when run with some other test(s)*". The flakiness of a Brittle comes from its reliance on some state that should be set up by other tests. When this precondition is missing, the Brittle exhibits a failing behavior. Table 3 illustrates additional definitions for tests that, although not flaky, play an important role in the behavior of OD tests. As introduced earlier, a **Polluter** (or *State-Polluting* test) is (1) "*a test that pollutes (i.e. modifies) the state shared across tests*". Gyori et al. [14]

Table 3. Classification of Order-Dependent Related Tests

Term	Definition	Source(s)
Polluter	(1) A test that pollutes (i.e. modifies) the shared state.	[14]
	(2) A test that pollutes the state on which a Victim depends.	[38]
Helper	A test whose logic (re)sets the state required for an Order-Dependent test to pass.	[38]
Cleaner	A test order that resets the state polluted by a polluter.	[38]
State-Setter	A test order that sets up the state for a brittle.	[38]

specify that if a test makes assumptions about the shared location, the resulting dependency can affect the reliability of its outcome. Shi et al. [38] provide a consistent definition (2), although they clarify that a Polluter can comprise multiple tests, as long as their combination causes a Victim to fail consistently. It is worth noting that both definitions suggest that a Polluter always causes its Victim to fail, raising a "*false alarm*". While this is the most common scenario [47], a polluter might also generate a state in which an OD test accidentally passes, masking a real fault in the code under test. Such failures are sometimes referred to as *missed alarms*[47] or *silent horrors*[45]. Shi et al. [38] also provide a definition for **Helpers**. These are commonly run in between OD tests to ensure that the state is properly cleaned or set up before their execution. In particular, the **Cleaners** reset the state previously modified by a Polluter, so that subsequent OD tests are not negatively affected by the dependency. Conversely, **State-Setters** implement logic that purposefully sets up the state required for a Brittle to pass.

Classification of NOD Tests As introduced earlier, a NOD test inconsistently passes and fails even for the same execution order. Given the erratic and usually infrequent manifestation of NOD flakiness, these tests are generally harder to identify and debug. Indeed, the inconsistency of the test outcomes is not simply attributable to the presence of a test order dependence. However, a recent work of Lam et al. [22] questions the adequacy of this definition, specifying that NOD flakiness can sometimes be affected by the execution order. As a result of this observation, the authors further refine the definition of a NOD tests, specifying that it (2) "*fails non-deterministically for at least one order (failure rate is neither 0% nor 100%)*". Depending on the failure rate associated to each execution order, a NOD test can be further classified into two groups. **Non-Deterministic Order-Dependent** (NDOD) show a significantly higher failure rate for at least one execution order. Since there exists an order for which the flakiness is much more likely to manifest, NDOD tests are characterized by an underlying order dependence. NDOD tests should not be confused with OD tests, because the latter always behave in a deterministic manner. On the other hand, **Non-Deterministic Order-Independent** (NDOI) tests are characterized by a similar failure rate for each possible execution order, thus they are more in line with the general idea of non order-dependent test. Again, this underlines the fact that Non-Deterministic tests and Flaky Tests should not be used as synonyms. Although NOD tests can be further classified according to the root causes of their flakiness, here we just focus on two further definitions that might generate confusion among researchers and practitioners. An **Implementation Dependent** (ID) test is defined by Lam et al. [23] as "*a test whose outcome depends on the implementation of a non-deterministic specification*". Therefore it can be identified as a NOD test whose flakiness is caused by wrong assumptions about the SUT, which unexpectedly behaves in inconsistent manners. Lastly, a **Smelly Test** is not necessarily flaky. The term "*smelly*" is commonly used for identifying any kind of a poorly designed test. We can depict a *test smell* as an anti-pattern that decreases the quality of the test suite and/or the code under test. The effects of a test smell can range from poor test code understandability up to pontentially missing severe bugs into the SUT. Several works [1, 2, 42] identify the presence of a test smell as a potential cause for test flakiness as well. In particular, Alshammari et al. [2] report specific classes of smells that can be commonly found in flaky tests. For instance, the *Mistery Guest* smell can be found in a test whose execution relies on some external resources. The underlying dependency can cause the test to exhibit a non-deterministic behavior, in that the availability of such resources can change over time.

4 Conclusions and Future Work

Motivated by the lack of a consistent vocabulary, we undertook a pragmatic scoping review of white and grey literature, based on which we reported a first analysis of definitions relative to flakiness concepts, as well as a first classification of flaky test types. This study provides a preliminary assessment of key concepts and evidences the need of establishing an agreed terminology, perhaps after having conducted a more extensive synthesis of current knowledge. We think that the study of causes and remedies to test flakiness must also link to other related research topics, such as the already mentioned early literature on replaying of concurrent tests [6] or even the several studies about the nature of bugs [8].

References

1. Ahmad, A., Leifler, O., Sandahl, K.: Empirical analysis of factors and their effect on test flakiness-practitioners' perceptions. arXiv preprint arXiv:1906.00673 (2019)
2. Alshammari, A., Morris, C., Hilton, M., Bell, J.: FlakeFlagger: Predicting flakiness without rerunning tests. In: Proc. ICSE Art. Ev. track. IEEE (2021)
3. Arksey, H., O'Malley, L.: Scoping studies: towards a methodological framework. International Journal of Social Research Methodology **8**(1), 19–32 (2005)
4. Barboni, M., Bertolino, A., De Angelis, G.: Supplemental Material: What we talk about when we talk about software test flakiness (Jun 2021). https://doi.org/10.5281/zenodo.5016745, `https://doi.org/10.5281/zenodo.5016745`
5. Bell, J., Legunsen, O., Hilton, M., Eloussi, L., Yung, T., Marinov, D.: DeFlaker: Automatically detecting flaky tests. In: Proc. ICSE. pp. 433–444. ACM (2018)

6. Carver, R.H., Tai, K.C.: Replay and testing for concurrent programs. IEEE Software **8**(2), 66–74 (1991)
7. Champier, C.: Flaky tests caused by a production bug: fix the flakiness, not the bug. Online on `medium.com` (Feb 2019)
8. Cotroneo, D., Grottke, M., Natella, R., Pietrantuono, R., Trivedi, K.S.: Fault triggers in open-source software: An experience report. In: Proc. ISSRE. pp. 178–187. IEEE (2013)
9. Dutta, S., Shi, A., Choudhary, R., Zhang, Z., Jain, A., Misailovic, S.: Detecting flaky tests in probabilistic and machine learning applications. In: Proc. ISSTA. pp. 211–224. ACM (2020)
10. Eck, M., Palomba, F., Castelluccio, M., Bacchelli, A.: Understanding flaky tests: The developer's perspective. In: Proc. ESEC/FSE. pp. 830–840. ACM (2019)
11. Eloussi, L.: Flaky tests (and how to avoid them). Online on `medium.com` (Sep 2016)
12. Fowler, M.: Eradicating non-determinism in tests (Apr 2011)
13. Groce, A., Holmes, J.: Practical automatic lightweight nondeterminism and flaky test detection and debugging for Python. In: Proc. QRS. pp. 188–195. IEEE (2020)
14. Gyori, A., Shi, A., Hariri, F., Marinov, D.: Reliable testing: Detecting state-polluting tests to prevent test dependency. In: Proc. ISSTA. pp. 223–233. ACM (2015)
15. King, T.M., Santiago, D., Phillips, J., Clarke, P.J.: Towards a bayesian network model for predicting flaky automated tests. In: Proc. QRS-C. pp. 100–107. IEEE (2018)
16. Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33**(2004), 1–26 (2004)
17. Kowalczyk, E., Nair, K., Gao, Z., Silberstein, L., Long, T., Memon, A.: Modeling and ranking flaky tests at Apple. In: Proc. ICSE-SEIP. pp. 110–119. ACM (2020)
18. Lam, W., Godefroid, P., Nath, S., Santhiar, A., Thummalapenta, S.: Root causing flaky tests in a large-scale industrial setting. In: Proc. ISSTA. pp. 101–111. ACM (2019)
19. Lam, W., Muşlu, K., Sajnani, H., Thummalapenta, S.: A study on the lifecycle of flaky tests. In: Proc. ICSE. pp. 1471–1482. ACM (2020)
20. Lam, W., Oei, R., Shi, A., Marinov, D., Xie, T.: iDFlakies: A framework for detecting and partially classifying flaky tests. In: Proc. ICST. pp. 312–322. IEEE (2019)
21. Lam, W., Shi, A., Oei, R., Zhang, S., Ernst, M.D., Xie, T.: Dependent-test-aware regression testing techniques. In: Proc. ISSTA. pp. 298–311. ACM (2020)
22. Lam, W., Winter, S., Astorga, A., Stodden, V., Marinov, D.: Understanding reproducibility and characteristics of flaky tests through test reruns in Java projects. In: Proc. ISSRE. pp. 403–413. IEEE (2020)
23. Lam, W., Winter, S., Wei, A., Xie, T., Marinov, D., Bell, J.: A large-scale longitudinal study of flaky tests. Proc. ACM on Programming Languages **4**(OOPSLA), 1–29 (2020)
24. Lee, B.: We have a flaky test problem. Online on `medium.com` (Nov 2019)
25. Liviu, S.: A machine learning solution for detecting and mitigating flaky tests. Online on `medium.com` (Oct 2019)
26. Luo, Q., Hariri, F., Eloussi, L., Marinov, D.: An empirical analysis of flaky tests. In: Proc. FSE. pp. 643–653. ACM (2014)
27. Machalica, M., Samylkin, A., Porth, M., Chandra, S.: Predictive test selection. In: Proc. ICSE-SEIP. pp. 91–100. IEEE (2019)
28. Malm, J., Causevic, A., Lisper, B., Eldh, S.: Automated analysis of flakiness-mitigating delays. In: Proc. AST. pp. 81–84. IEEE (2020)

29. Micco, J.: Flaky tests at Google and how we mitigate them (May 2016)
30. Munn, Z., Peters, M.D., Stern, C., Tufanaru, C., McArthur, A., Aromataris, E.: Systematic review or scoping review? guidance for authors when choosing between a systematic or scoping review approach. BMC medical research methodology **18**(1), 1–7 (2018)
31. Otrebski, K.: Flaky tests. Online on `medium.com` (Apr 2018)
32. Palmer, J.: Test flakiness – methods for identifying and dealing with flaky tests. Online on `medium.com` (Nov 2019)
33. Parry, O., Kapfhammer, G.M., Hilton, M., McMinn, P.: Flake it'till you make it: Using automated repair to induce and fix latent test flakiness. In: Proc. ICSE Workshops. pp. 11–12. ACM (2020)
34. Presler-Marshall, K., Horton, E., Heckman, S., Stolee, K.: Wait, wait. no, tell me. analyzing selenium configuration effects on test flakiness. In: Proc. Wksp AST. pp. 7–13. IEEE (2019)
35. Rahman, M.T., Rigby, P.C.: The impact of failing, flaky, and high failure tests on the number of crash reports associated with Firefox builds. In: Proc. ESEC/FSE. pp. 857–862. ACM (2018)
36. Shi, A., Bell, J., Marinov, D.: Mitigating the effects of flaky tests on mutation testing. In: Proc. ISSTA. pp. 112–122. ACM (2019)
37. Shi, A., Gyori, A., Legunsen, O., Marinov, D.: Detecting assumptions on deterministic implementations of non-deterministic specifications. In: Proc. ICST. pp. 80–90. IEEE (2016)
38. Shi, A., Lam, W., Oei, R., Xie, T., Marinov, D.: iFixFlakies: A framework for automatically fixing order-dependent flaky tests. In: Proc. ESEC/FSE. pp. 545–555. ACM (2019)
39. Silva, D., Teixeira, L., d'Amorim, M.: Shake it! detecting flaky tests caused by concurrency with Shaker. In: Proc. ICSME. pp. 301–311. IEEE (2020)
40. Słapiński, M.: What is flakiness and how we deal with it. Online on `medium.com` (Feb 2020)
41. Stosik, D.: Dealing with flaky tests. Online on `medium.com` (Nov 2019)
42. Strandberg, P.E., Ostrand, T.J., Weyuker, E.J., Afzal, W., Sundmark, D.: Intermittently failing tests in the embedded systems domain. In: Proc. ISSTA. pp. 337–348. ACM (2020)
43. Terragni, V., Salza, P., Ferrucci, F.: A container-based infrastructure for fuzzy-driven root causing of flaky tests. In: Proc. ICSE-NIER. pp. 69–72. IEEE (2020)
44. Thorve, S., Sreshtha, C., Meng, N.: An empirical study of flaky tests in android apps. In: Proc. ICSME. pp. 534–538. IEEE (2018)
45. Vahabzadeh, A., Fard, A.M., Mesbah, A.: An empirical study of bugs in test code. In: Proc. ICSME. pp. 101–110. IEEE (2015)
46. Waterloo, M., Person, S., Elbaum, S.: Test analysis: Searching for faults in tests (n). In: Proc. ASE. IEEE (Nov 2015)
47. Zhang, S., Jalali, D., Wuttke, J., Muşlu, K., Lam, W., Ernst, M.D., Notkin, D.: Empirically revisiting the test independence assumption. In: Proc. ISSTA. pp. 385–396. ACM (2014)
48. Ziftci, C., Cavalcanti, D.: De-Flake your tests: Automatically locating root causes of flaky tests in code at Google. In: Proc. ICSME. pp. 736–745. IEEE (2020)
49. Zolfaghari, B., Parizi, R.M., Srivastava, G., Hailemariam, Y.: Root causing, detecting, and fixing flaky tests: State of the art and future roadmap. Software: Practice and Experience (2020)