Flaky tests

SE Radio episode 572

Mario Junquera Rojas – UO287557 Lucía Ruiz Núñez - UO289267 Didier Yamil Reyes Castro - UO286866

Introduction

Have you ever experienced the frustration of test failures messing up with your software development process? If so, you're not alone.

Flaky tests are the Schrodinger cat of tests, this means that they can pass or not without changing a line of code and until you don't run them you don't know what will happen. This is truly a problem because we cannot apply continuous integration nor development as they need tests to verify that all works.

Types of Flaky Test

Flaky tests can be categorized into two types: order dependent and non-order dependent.

- Order dependent tests exhibit flakiness by failing or passing inconsistently when executed in a different sequence than originally intended. For instance, if a set of tests passes when executed in a specific order but fails when reordered, they are deemed flaky.
- Non-order dependent tests are unaffected by the sequence of execution. In such cases, flakiness may stem from timing issues, where the test only succeeds during specific intervals due to functionality constraints within a given time frame. Additionally, internal system components such as CPU, memory usage, or file system access can also contribute to test flakiness.

Common Sources of Flaky Tests

- Physical Sources: when we talk about physical sources, we're primarily referring to the hardware components of a system. However, can hardware impact testing? Yes, when crafting tests, we often consider the memory and CPU specifications. But what if these resources are altered - decreased or increased? Such changes can introduce flakiness into our tests, as resource usage may fluctuate from one test run to another.
- Software Sources: on the software side, several factors contribute to test flakiness:
  - Timing Issues: Consider testing the user interface (UI) of a Wikidata application, particularly the question generation feature. Sometimes, a test may pass if questions are swiftly generated, while others may fail due to delayed responses from the Wikidata API.
  - Database and File System Access: Accessing information from databases or file systems can occasionally result in delays, leading to flakiness in tests reliant on such data retrieval.
  - Date and Time Considerations: These are distinct from timing issues. Imagine a test suite validating the arrival date and time of a flight. This test is inherently tied to a specific time zone. But what happens if the test is conducted on a

server located in a different time zone, such as Japan? Mismatched time zones can skew test results unpredictably.

Identification and Response

When it comes to tackling flaky tests, one effective approach is to isolate the suspect test and rerun it multiple times in a controlled environment. If it passes in isolation, we can gradually reintroduce other tests from the suite to see if any interactions between them trigger flakiness.

Thankfully, there are tools available that streamline this process by automating the rerunning of tests and flagging those that exhibit flakiness. This helps streamline the identification of problematic tests within a suite.

Once a flaky test has been identified, the next step involves delving into its execution environment. Tools like Datadog can be invaluable in this regard, providing insights into the underlying factors contributing to the flakiness. Armed with this information, we can take proactive measures such as temporarily halting the test from running in continuous integration pipelines. Additionally, exploring alternatives like mocking third-party services can help mitigate dependencies that may be exacerbating the flakiness.

AI and Flaky Tests

The role of artificial intelligence (AI) is increasingly being considered. Gregory Kapfhammer sheds light on how AI, particularly supervised ones can help detect flaky test.
Kapfhammer's supervised learning approach shows promise. It analyzes code patterns to identify flakiness, albeit with probabilistic outcomes.

However, AI models, including those for flaky test detection, offer insights with a degree of uncertainty. Despite advancements, there's inherent ambiguity in their conclusions.

An example on how AI works is by using abstract syntax trees (AST) to pinpoint potential flakiness, particularly in complex code structures. It's also adept at detecting runtime flakiness, such as memory issues or filesystem misuse.

Tips

While writing test cases, we probably introduce flakiness in those ones, so the following are some tips to avoid them:

- The most important one: Running tests in a random order. This is a way of tackling order-dependent test.
- About writing test cases: They must be as simple as possible and follow the SRP. Also, conditionals or loops in since they mean that a test can have different paths to follow.
- Tied with previous one, we must have the best possible set up and tear down methods associated with all tests, especially those test cases that are flaky. Obviously, there’s a tradeoff here! We must try to find the right balance between clearing out enough of the shared state of tests so that we don’t have flakiness, but maybe allowing some shared state as long as it doesn’t negatively influence the outcomes of test cases.