Five causes for flaky tests happen and what you can do about them

Flaky tests are non-deterministic tests in your test suite. They may be intermittently passing or failing, making test results unreliable. Developer productivity is heavily dependent on accurate test results and goes down if flaky tests increase. With multiple, unrelated commits causing similar errors, change management can quickly turn into a nightmare!

Legitimate issues may get ignored due to a high number of false positives. Repetitive work is required to determine if bugs exist at all and whether they are present due to an error in code or an issue with the test itself. Actual fixing of problems can take a back seat and cause user dissatisfaction with bugs ending up in production.

Let us look at five common causes for flaky tests showing up in your build pipeline and what you can do about them.

Shared state resources between tests – One of the common reasons why tests do not find bugs is concurrency. They occur because developers may have made incorrect assumptions about the ordering of operations between threads. One test thread might be assuming a state for shared resources like data or memory.

For example, test 2 might assume test 1 passes and use test 1's output as an input for itself. Or test 2 might assume that test 1 leaves a data variable in state x, but test 1 may not always do that – causing test 2 to fail. Tests can also be flaky if they do not correctly acquire and release shared resources between them.

What can you do about it?

- Use synchronization blocks between tests.
- Change the test to accept a wider range of behaviors.
- Remove dependencies between tests.
- Explicitly set static variables to their default value.
- Use resource pools - your tests can acquire and return resources to the pool.

Unreliable 3rd party APIs – Decreased control of your test environment increases the chances of test unpredictability. Flaky tests can occur when your test suite is dependent on unreliable third-party APIs or functionality maintained by another team. These tests may intermittently fail due to third-party system errors, unreliable network connections, or third-party contract changes.

What can you do about it?

- Use test stubs or test doubles to replace the third-party dependency. Your regular tests can talk to the double instead of the external source.
- Test doubles will not detect API contract changes. You will need to develop a separate suite of integration contract tests for this.
- Contract tests can be run separately and need not break the build the same as other tests. They can be run less frequently and be actioned independently of other bugs.
- Communicate with the third-party provider to discuss the impact of changes made by them on your system.

Infrastructure issues – Test infrastructure failure is one of the common causes for flaky tests. These include network outages, database issues, Continuous Integration Node Failure, etc.

What can you do about it?

- These issues are typically easier to spot than others. Your debugging process can check these first before attempting to find other causes.
- Write fewer end-to-end tests and more unit tests.
- Run tests on real devices instead of emulators or simulators.

Flaky UI tests – UI tests are used to test visual logic, browser compatibility, animation, etc. Since they start at the browser level, they can be very flaky due to a variety of reasons – from missing HTML elements, cookie changes, etc. to actual system issues. If you visualize your test suite as a pyramid, UI tests are at the top. They should only occupy a small portion of your test portfolio because they are brittle, expensive to maintain, and time-consuming to run.

What can you do about it?

- Don't use UI tests to test back-end logic.
- Capture the network layer using Chrome DevTools Protocol(CDP). CDP allows for tools to inspect, debug, and profile Chromium, Chrome, and other Blink-based browsers

Badly written tests – Not following good test writing practices can result in a large number of flaky tests in your pipeline. Some common mistakes include:

- Not adopting a testing framework from the start.
- Caching data. Over time, cached data may become stale affecting test results.
- Using random number generators without accounting for the full range of possibilities.
- Using floating-point operations without paying attention to underflows and overflows.
- Making assumptions about the order of elements in an unordered collection.
- Using sleep statements to make your test wait for a state change. Sleep statements are imprecise and one of the biggest causes of flaky tests. It is better to replace them with the waitFor() function0.

What can you do about it?

- Treat automation testing like any other software development effort. Make testing a shared responsibility between developers and analysts.
- Use tools to monitor test flakiness. If the flakiness is too high, the tool can quarantine the test, (removing it from the critical path) and help resolve issues faster.
- Start all tests in a known state.
- Avoid hardcoding test data.

Is there a way to eliminate flaky tests completely?

The unfortunate answer is, no you cannot. Even high-performing teams like Google have reported flakiness in 16% of their test suite. The best way to deal with the issue is by

adopting best practices in testing and using the latest tools. You can contact founders@mergequeue.com to get an invite to our private beta group for our Flaky Test Manager product. Flaky Test Manager is an innovative and up-to-date testing tool that is guaranteed to make your development process much smoother!