TechWell Insights

Defensive Design Strategies to Prevent Flaky Tests

By Steve Berczuk (/users/steve-berczuk) - April 29, 2020

Agile code (https://www.stickyminds.com/better-software-magazine-article/agile-code-agile-teams) is at the heart of agile software development. It's not enough to embrace change in the requirements process; you need to be able to deliver incremental changes along the way, since agile methods deal with uncertainty in the product space by enabling stakeholders to inspect and adapt. Without working code that can be delivered into production quickly, it's hard to inspect things in a meaningful way.

There are many elements to agile code. Designing with patterns to allow for flexibility with minimal overhead is essential, but not enough. You also need testing in order to get feedback (/techwell-insights/2020/04/achieve-repeatable-builds-continuous-integration%5D).

More testing isn't always better testing, though. If your tests are unreliable, they can cause more harm than good. "Flaky tests" aren't as rare as we'd like to think, and there are a few things you can keep in mind to avoid them and help your development lifecycle move more smoothly.

Flaky tests are tests that fail intermittently, and whose failures don't really provide useful indications of significant errors in the product code. Flaky tests could be the result of issues in the code, but more often they are the result of assumptions in the test code that lead to non-relatable results. Common examples are UI tests that rely on identifying elements based on positions, or data-driven tests that make assumptions about the initial state of the data without verifying it.

While integration tests are often the most likely to exhibit flakiness, unit tests are not immune. Unit tests based on time computations can fail on time zone transitions, or if a legacy test is written with the assumption of a future date that is fixed.

There are many reasons that tests can fail intermittently, and some can be easily avoided by applying good defensive design strategies. Sometimes the problem is that the test is too simple—for example, comparing a response to an API call as a string, when you have no control over order or even whitespace. Instead, compare fields in a structured object, checking only for the fields you expect.

Sometimes the issue is testing something irrelevant. Comparing all the fields in a response rather than just the ones that matter to the client that you are testing can lead to test failures that don't imply code failures.

Stateful integration tests can fail when you make assumptions about the data that you can't verify. Ideally, you would start with a known external system state, but if that's not possible, consider structuring tests so that the query can examine the current state, make a change, and then examine the new state.

While one can argue that tests that provide more noise than signal should just be skipped, that is not always a good practice. Even very simple tests (https://blog.berczuk.com/2009/05/really-dumb-tests.html) that provide valuable insight can be flaky if not written well.

Test code *is* code, and it should be treated with the same level of good design that you treat production code. It should be adaptable and only make assumptions that it verifies. By making this a practice, you can keep all of your code more agile.

Tags:
agile (/keywords/agile), programming (/keywords/programming), test design (/keywords/test-design), test management (/keywords/test-management), test techniques (/keywords/test-techniques), testing (/keywords/testing), flaky tests (/other-keywords/flaky-tests)

Up Next

Build Better Teams by Finding Hidden Talents (/techwell-insights/2020/04/build-better-teams-finding-hidden-talents)
April 28, 2020

1 comment


(/users/jason-rudolph) Jason Rudolph (/users/jason-rudolph)

If your tests are unreliable, they can cause more harm than good.

I totally agree. Once the build is regularly failing due to flaky tests, it's easy to lose trust in the test suite. Before long, you find yourself clicking "rebuild" without even looking at the failing test to see if it might be a legitimate failure.

We got hit pretty hard by this while working on Atom (https://github.com/search?q=author:jasonrudolph+flaky+org%3Aatom&utf8=%E2%9C%93). It was a lot of manual work trying to keep track of which tests were flaky, and we never felt like we had a reliable assessment of which tests were flaky and just how flaky they were.

I'm hoping that tools like https://buildpulse.io (https://buildpulse.io) (which I'm currently working on) can make it easier for teams to keep their test suite healthy. By **automatically detecting flaky tests** and showing their failure rate over time, everyone gets a shared understanding of how/whether flaky tests are impacting the project. And by **identifying the most disruptive flaky tests**, teams can see exactly where to start in order to have the most impact on improving the reliability of the test suite.

June 28, 2020 - 7:33pm

Login (/user/login?destination=node/199769%23comment-form) or Join (/user/register?destination=node/199769%23comment-form) to add your comment

About the Author

(/users/steve-berczuk)

Steve Berczuk (/users/steve-berczuk)

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration (http://www.stickyminds.com/s.asp?F=S541_BOOK_4)*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com (mailto:steve@berczuk.com) or visit berczuk.com (http://www.berczuk.com/) and follow his blog at blog.berczuk.com. (http://blog.berczuk.com)