**What Flaky Tests Can Tell You**

**Joshua Grant,** [**Joshua.m.grant@gmail.com**](mailto:Joshua.m.grant@gmail.com)

**For: Sticky Minds**

Imagine this: You are a test developer who creates automated end-to-end tests for a web application. You use good technology in the Selenium WebDriver. You follow good coding practices like using page objects to separate the tests from application logic. You make use of virtualization and use adequately powered machine hardware. You also understand how end-to-end automation is a team effort and so you work with application developers, testers and managers in concert to get the most value out of your work. And yet for all your good work you still have a recurring problem: flaky tests. These are tests that pass or fail unexpectedly for reasons that appear random. Flaky tests become even worse as test suites grow and more areas of an application are covered. Eventually there comes a temptation to throw up your hands and say "This is stupid!" and throw the whole end-to-end test suite into the trash bin.

But are flaky tests really a problem?

It can easy to shrug off flaky tests and use them to discredit automated end-to-end testing. But flaky tests can be useful for learning more about an application and about your team dynamic. I'll provide some technical and human examples of where flaky tests can be helpful in software testing efforts.

The Technical Side

From a technical standpoint, there are a few common sources of flakiness in WebDriver-based tests. One of the main culprits is synchronization (or a lack thereof). Web applications have many layers that affect performance, including network connection speed, HTTP handling, source rendering and computer processing resources. As a result, some operations may vary slightly in timing during different runs of end-to-end scenarios. A button may not appear quickly enough or a dialog box may not disappear fast enough for an automated script to complete without unexpected failures. The solution is to put wait statements to synchronize script steps with the application. This might seem like a hack to avoid flakiness, but it also may be an oracle of performance issues in your application. If some areas consistently need more waits or longer waits it could be an indication of poor performance, particularly client-side performance, in those areas. At one point on one team I worked with, there was one set of automated end-to-end tests that seemed to fail inconsistently all the time but related to the same feature. When I talked to developers, it turned out that area had some front-end issues due to some bad coding practices. Flaky tests picked up on this problem if indirectly.

Another problem I've seen producing flaky tests is from accidental load testing. As end-to-end automated test suites grow, the number of lines of test code grows but also there are more tests being executed against the application under test. In turn this usually means that test suites are re-organized to run in parallel or concurrently to help cut down on test runtime. While helpful for testers and developers, this can also have a side-effect of putting large loads on your application, creating an unintended load test. Automated end-to-end tests that run perfectly fine in series might get flaky when run concurrently. In one case I saw on a project I was working on, some tests were run initially individually were working just fine but had some problems when we first tried running them in parallel, with a few (seemingly random) failures. After some debugging, one of my team mates found that when run in parallel, our tests would try all to login with the same admin user instantly when tests started, resulting in around eight simultaneous logins by the same user. The application was not prepared for this and we found out the hard way, but this flakiness was beneficial and helped us design better tests.

The Human Side

Writing automated end-to-end tests is a testing activity and so it is important to think about them from a testing perspective. One great use of flaky tests is as barometer of teamwork and team communication. One challenge I've encountered several times is getting team members to take interest in end-to-end test results. Since flaky tests will sometimes appear to fail and other times appear to pass, interested team members - that is, people who are actually looking at test results - will ask about them. Even if the answer for flakiness is "They're flaky", this is often a good place to start conversations about testing, quality, and automation approaches. In my experience, if several tests are flaky and no one is asking about it your team either is not getting information about test results or is not interested. Your team not getting information properly is a completely solvable problem, but one that is sometimes tricky to identify. It was not until I talked with one of the developers I work with on my current team that I found out application developers were unable understand how to interpret our test output (from our continuous integration server), and that issue partially arose because he was interested in why some of our tests were failing. If your team is completely indifferent to end-to-end automated test results completely, that is also solvable but requires a bit more creativity.

Following along with gauging interest, flaky tests might also be able to tell you about "test results fatigue", a condition where teams are so inundated with unreliable test results that they begin ignoring end-to-end automated tests. What starts off as a promising testing effort might eventually be ruined when team members ignore some flaky tests, then all tests related to the flaky tests, then effectively all tests. Test result fatigue is a scourge that can kill the benefit of automation, and a prime cause of test fatigue is flaky tests. Watching how your team reacts over time to flaky tests might give you insight to how much they're invested in automation over time, even past the honeymoon period of using any new tool. It might also give you some insight to how engaged overall your team is with a project at any given time.

Lastly, consider how automated end-to-end tests are being used in the context of your team or application. In a team or organization that practices continuous deployment, passing automated end-to-end tests may be a *requirement* for product builds or releases. Flaky tests that are needlessly halting builds or releases are a serious problem that need attention. In this case automated end-to-end tests are de facto acceptance tests (or if one prefers, rejection checks) and should be treated as such. Teams that use automated end-to-end tests as a regression testing approach - checking for known types of bugs before release - can view flaky tests differently. Here tests can be interpreted by people and in turn can be interpreted accordingly. Both approaches are viable the important thing is to understand *which approach* your team uses.

Flaky tests can help answer the question of what your team is trying to accomplish overall with end-to-end automation and to further identify how to make your software better.