SauceLabs

WHITE PAPER

The Four Keys to Achieving Parallelization in Automated Testing

Introduction

Agile development has reached a crossroads. Keenly aware of the need to deliver high-quality web and mobile applications at speed, most organizations have adopted modern agile development methodologies with the expectation that doing so will drive faster release cycles, improve functional quality, and, ultimately, lead to better customer experiences.

Yet, for many organizations, release velocity has stalled, if not declined. Forrester reported in 2018 that only 27 percent of organizations released software at least once per month. As of 2021, that number had inched up only to 31.3 percent – hardly a momentous gain. Clearly, our collective efforts to deliver better software faster have hit a roadblock.

Those that have not yet taken advantage of automated testing are missing out on crucial opportunities to improve software release velocity.

Table of Contents

SauceLabs

Parallelization: The Only Path Forward

Not only that, but they must also learn to take full advantage of parallelization, which requires test automation.

At some point in their automated testing journeys, most organizations encounter the same problem: as the number of tests in a suite starts to grow, the suite starts taking too long to run. Testing is the foundation of agile development, so once your suites start taking too long to run, your efforts to deliver quality at speed eventually stall out.

Fortunately, there's a clear solution: parallelization. Take the hypothetical example of a suite of 100 tests, each of which takes two minutes to run. If you can run those tests in parallel, you'll complete your entire suite of tests in just two minutes, enabling your developers to quickly get the feedback they need. If, on the other hand, you're unable to leverage parallelization, that same suite of tests will take more than three hours to run, while your developers wait unproductively.

Leveraging parallelization is the only way to successfully implement automated testing. But most organizations immediately run into a number of hurdles as they attempt to scale their automation efforts through parallel testing. Let's take a look at how to overcome those hurdles by examining the four mandatory requirements for effective parallel testing.

Requirement #1: Run Atomic Tests

The most powerful and effective strategy for achieving parallelization is to run atomic tests. Atomic tests assess just one single application feature. So, rather than scripting a single test to validate that the login page loads, the user name field displays, the password field displays, the logout displays, and items can be added to a cart, a team leveraging atomicity would design five separate tests that validate each of those functions individually.
[For more on atomic tests, check out this Sauce Labs video.]

The benefits of atomic tests are many. When I conduct automated testing workshops, I often open by asking attendees to guess which of the following test suites, each covering the same exact application features, will execute faster: a test suite featuring 18 long-flow, end-to-end tests, or a suite using 180 atomic tests. The answer, to the surprise of most, is the suite featuring 180 atomic tests. In fact, when I run live demos using this exact scenario, the suite featuring 180 atomic tests typically executes 8 times faster than the suite with 18 long-flow tests!

Most organizations mistakenly assume that running longer tests in smaller quantities is a faster approach than running atomic tests in larger quantities. Thus, they attempt to combat longer-than-desired test run times by adding more validations to their tests. Doing so only adds fuel to the fire. No matter how much parallelization you apply, your test execution time will only be as fast as the slowest test in your suite. So, if you have a suite of 30 tests, 29 of which take 2 minutes to execute, and one of which takes 30 minutes to execute, you'll have to wait the full 30 minutes to get the results of every test in that suite.

Atomicity Drives Stability

Suites that leverage atomic tests are far more stable than those that don't. After all, every single validation request you add to a test is another chance for something to go wrong. Atomic tests are also

far easier to debug when a test does fail. Because atomic tests execute much faster than longer tests, developers are getting feedback on code they literally just wrote, making it exponentially easier to go back and fix.

Moreover, since atomic tests focus on just one specific piece of application functionality, there's no ambiguity about what's broken in the event of a failed test. It can only be that one thing, and developers don't have to waste precious time rooting around for the source of the failure. By comparison, when a non-atomic test fails, developers get no feedback on features beyond the point of the failure. So, if you're testing 50 different elements of functionality within a single test, and the failure occurs at element 10, the remaining 40 elements go untested.

Requirement #2: Keep Tests Autonomous

An autonomous test is one that can run completely independent of all the other tests in your suite. Many teams make the mistake of designing a multi-threaded process in which one test cannot successfully execute until its predecessor has done the same.

This means, for example, that in order to execute a test validating the efficacy of your checkout function, you first have to successfully execute tests for all of the functions that precede it in the application workflow. It also means that once one test fails, all of the other dependent tests will fail as well. This type of co-dependent approach is a non-starter for effective parallel testing. Instead, make sure you design your tests such that they can all run entirely on their own and in any order necessary.

Requirement #3: Correctly Manage Test Data

To effectively implement parallel testing, you must be able to control your test data. This is difficult to do even in the best of circumstances, as it depends on more than just your automation engineers to execute. But it's especially difficult to do if you're relying on traditional hard-coded test data, the static nature of which is a poor fit for the dynamic nature of automated testing.

Instead, the best strategy is to leverage what's known as just-in-time data. With a just-in-time approach, you create test data, utilize it for a given automated test suite, and then destroy it at the end of the test. This cuts down on complexity and ensures that data from a previously executed test doesn't muddy the results of your current test.

Requirement #4: Avoid The Static Keyword

The final mandatory requirement for effective parallel testing is to avoid applying the "static" keyword to the WebDriver instance you're managing in your test scripts. Using the "static" keyword is the fastest way to kill your parallelization dreams.

Identifying the WebDriver as "static" effectively ties the WebDriver instance to the definition of the class, not instances of the class, which means that there can only be one instance of WebDriver attached to (and shared between) all of the tests in your suite. It's like telling all the cars in the world that they have to share a single steering wheel!

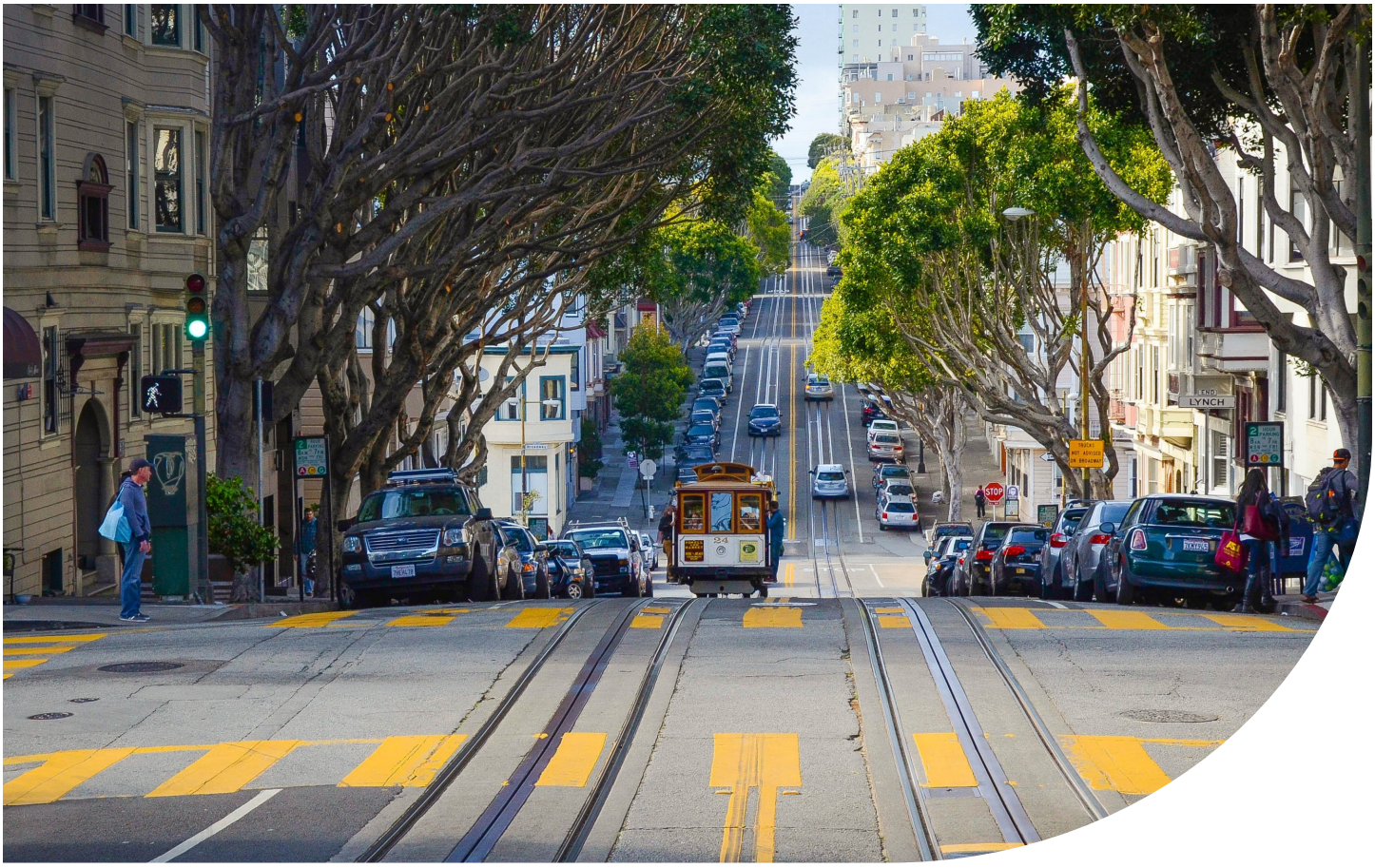
Some testers will work around this roadblock by people choosing to “fork” the JVM process for each test in their suite, thus creating a new instance of the entire test suite for every test. To extend the car metaphor, this is like recognizing that your cars all share the same steering wheel and compensating for it by creating a new planet Earth for each car you want to put on the road. Not exactly the most efficient workaround. Though use of the “static” keyword is technically acceptable in certain situations, as a general rule, the best approach is to simply avoid it.

Putting It All Together: Excellence Through Automation

As you move through implementation of each of these critical requirements, it’s important to always keep the end goal in mind. That end goal, of course, is happy customers. In the modern age of digital business, the best and fastest way to turn prospects into customers (and customers into loyal advocates) is by rapidly delivering high-quality web and mobile applications. If you can achieve parallelization in automated testing, that end goal is well within reach.

About the Author

Nikolay Advolodkin is a Senior Solutions Architect at Sauce Labs. He has an extensive background in software testing, quality assurance and test automation as the CEO and Test Automation Instructor at UltimateQA. com, a training site full of videos and resources covering the gamut of testing topics and technologies.



About Sauce Labs

Sauce Labs is the leading provider of continuous test and error reporting solutions that gives companies confidence to develop, deliver and update high quality software at speed. The Sauce Labs Continuous Testing Cloud identifies quality signals in development and production, accelerating the ability to release and update web and mobile applications that look, function and perform exactly as they should on every browser, operating system and device, every single time. Sauce Labs is a privately held company funded by TPG, Salesforce Ventures, IVP, Adams Street Partners, and Riverwood Capital.

For more information, please visit
→ saucelabs.com

saucelabs.com/sign-up

FREE TRIAL

SAUCE LABS INC. - HQ 450 Sansome Street, 9th Floor, San Francisco, Ca Usa 94111

SauceLabs