SauceLabs

WHITE PAPER

Making the Move to Automated Testing

LEARN MORE AT SAUCELABS.COM

UPDATED MARCH 2022

Introduction

If you work in DevOps or software QA, you probably already know why automated software testing is essential for quality control to keep pace with continuous delivery. Automated testing delivers faster results and more thorough coverage. It also facilitates shift-left testing, which enables earlier detection of bugs. And it's a key step toward increasing IT automation as a whole – a goal that nearly nine out of ten enterprises are focusing on as of 2022.

Yet the question of how to approach automated testing remains difficult for many QA teams to solve. A variety of obstacles make it challenging to begin automating software tests or to increase the extent of test automation beyond the basics.

Table of Contents

For example, some software delivery teams lack previous experience with test automation and don't know where to start. Others may be aware of the basic principles, but struggle to decide which testing frameworks to use or which types of tests to automate first.

It's understandable, on the basis of these and other challenges, why the implementation of test automation has been slow at many organizations. Five years ago, fewer than one-third of businesses were automating software testing in any way. However, test automation adoption is likely to continue increasing as more recent data projects that the test automation market will increase in the next few years.

So, if your team has yet to embrace automated testing, you're not alone. But if you want to move faster, improve software quality, and scale tests to support more applications and environments, you need to find ways to overcome the hurdles and put automated testing into practice.

That's why we've prepared this eBook: to help DevOps teams, QA engineers, and testing professionals learn how to adopt automated testing. By providing clear answers to the most common questions that engineers face when they attempt to implement automated testing, the following pages offer step-by-step guidance for fully capitalizing on the many benefits that test automation stands to offer.

Question 1: Which Testing Model Should We Use?

There are three main testing models that organizations typically consider when making the move to automated testing.

Unit Tests
Unit tests, which are typically used to test whether a specific piece of code or function performs as expected, are the most basic form of tests. They are also typically the easiest to write. You can find a simple example of a unit test and a discussion of the limits of unit testing here.

Test-Driven Development
A second common testing model is test-driven development, or TDD. Using this strategy, organizations write tests to check whether part of an application or service is sufficient for a specific set of use cases. Unlike unit testing, TDD makes it possible to verify the functionality of software. However, TDD's effectiveness is limited to the use cases that TDD tests support. Not all facets of application functionality and user experience can be represented with TDD.

Behavior-Driven Testing
The third and most sophisticated testing model is behavior-driven testing, or BDD. BDD takes TDD to the next level by testing overall application behavior rather than focusing on specific parts of an application. BDD provides for the broadest range of test coverage, and it requires the greatest level of investment in designing and writing tests.

The Best Test Model Is All of Them
Because unit testing, TDD, and BDD represent distinct approaches to test design and implementation, it can be easy to assume that you have to choose one testing model, and one only.

In reality, these are not mutually exclusive testing models. Most organizations that perform a significant number of automated tests will use each of these models at different stages of the delivery pipeline:

- Unit testing works best during the code integration stage.
- TDD is useful for pre-production testing or the testing of individual microservices.
- BDD is ideal for testing just prior to release, as well as in production.

Question 2: Which Test Automation Framework Works Best?

There is no shortage of choice when selecting a software framework for writing and executing automated tests. Deciding which automated testing framework or frameworks to adopt is therefore one of the most intimidating challenges that organizations face when beginning a test automation strategy.

You do not, of course, have to limit yourself to one testing framework. It's common to use multiple frameworks at once for different purposes; for example, you might use one framework for non-mobile applications and another for mobile.

An exhaustive list of automated testing frameworks is too extensive to include here, but the following are the most popular options.

Selenium
Selenium is an open source automated testing framework designed by QA engineers to test applications within multiple types and versions of Web browsers. Selenium is the industry-standard, W3C-recommended automated testing framework.

The current version of Selenium is Selenium 4, which introduces a variety of new features and benefits compared to earlier releases. For the details, check out our guide to Selenium 4.

Appium
Appium, which was derived from Selenium, extends automated testing functionality for QA engineers to mobile applications, which Selenium does not directly support. Appium is almost certainly the most popular automated testing framework for mobile. Appium supports Android and macOS applications.

We mention some alternatives to Appium further down this list, but for a longer discussion, check out the comparison of mobile test automation frameworks on the Sauce Labs website.

Selendroid
Selendroid is another Selenium-like testing framework designed for mobile applications. Selendroid did not always support macOS, which has historically made it less popular than Appium. Today, however, Selendroid supports macOS as well as Android. Selendroid also offers the advantage of supporting more versions of the Android API than Appium, which can be an attractive feature if you need to test older Android apps.

XCUITest
As one component of Apple's XCode development system, XCUITest is designed for programmers seeking to test iOS applications during development. It supports a broad range of tests for macOS environments. For more details on XCUITest, refer to Sauce Labs's free whitepaper, Beyond Appium: Testing Using Espresso and XCUITest.

Espresso
Espresso is to Android applications what XCUITest is to macOS apps. Developed by Google, Espresso is designed for Android UI testing by developers. Refer to our Beyond Appium white paper for further information on using Espresso.

Mocha
Mocha is a popular testing framework for JavaScript applications. As such, it can be a useful resource for Web or mobile applications that rely on JavaScript.

Jasmine
Jasmine is another popular framework for JavaScript testing. For details on the differences between Jasmine and Mocha, and why you might choose one framework over the other, this article is a helpful resource.

Question 3: Which Test Grid Infrastructure Do I Need?

When it comes to actually running automated tests, you need test grid infrastructure to host them. A test grid is a cluster of physical or virtual machines that are configured to emulate different combinations of operating systems and browsers. In some cases, especially in the context of mobile testing, test grids may also include real devices that provide a non-emulated testing environment.

The approach that you take to obtaining a test grid can significantly impact the overall cost and maintenance burden associated with testing.

There are two main types of test grid architectures: on-premise and cloud-based. An on-premise test grid can offer some advantages when it comes to controlling data, but it can be expensive to set up and maintain. This is especially true if you want to include real mobile devices, of which there are tens of thousands of distinct models in existence. Even if you choose to include just a fraction of those mobile devices, acquiring and maintaining them would be challenging.

The alternative is a cloud-based test grid. A cloud-based grid eliminates the need for users to set up and maintain test infrastructure themselves, and it provides access to a vastly larger combination of software and hardware environments than most organizations could manage using an on-premise grid.

In addition, cloud-based grids offer the advantage of virtually unlimited scalability: whereas the number of tests that can be run in parallel on an on-premise grid is constricted by the size of the host infrastructure, cloud-based testing services have a much greater ability to scale.

For a complete discussion of the respective advantages and disadvantages of on-premise vs. cloud-based test grids, refer to the Sauce Labs whitepaper "Selenium Grid: Build vs. Buy."

Question 4: How Much Should You Automate?

One basic tenet of automated testing that can be easy for software delivery teams to overlook is that not all tests need to be automated. Even the most advanced QA operations will sometimes involve manual tests, for reasons detailed in the following section.

So, instead of striving for the unrealistic goal of fully automating all of your organization's software testing, set a feasible target for the portion of tests that you aim to automate. If you are new to automated testing, you might start by migrating just ten percent of your tests from manual to automated. If you're already doing some automated testing but want to do more, you can increase that percentage.

A healthy final target for the typical organization is to automate about 60 to 70 percent of tests.

Question 5: Which Tests Should Remain Manual?

When you set a target for the percentage of tests to automate, you should also determine which specific types of tests to automate, and which to keep performing manually.

As we discuss in more detail on the Sauce Labs blog, the main use case for manual testing is situations where you need nuanced insight into how users experience an application. If you are unsure which features users like, for example, or you need to test accessibility functionality, manual tests that allow you to evaluate individual users tend to work best.

For tests that don't focus on individual user experiences, however, automation is typically the way to go.

Question 6: Which Tests Should You Automate First?

Taking the first step into automated testing entails deciding not just which types of tests to automate and which to perform manually, but also which ones to automate first. Even if your eventual goal is to automate more than half of your tests, you cannot realistically convert all of them to an automated framework overnight.

Instead, start with tests that can be easily automated and whose automation delivers the most value. Integration, regression, user acceptance testing (UAT), and nightly build tests tend to fall within this category, since they are easy to write and take a long time to perform manually.

It also makes sense to prioritize the automation of tests that run most frequently. One of the main benefits of automated testing is that once you write tests, you can reuse them without limit. The more often you have to run a test, the greater the benefit you will obtain by automating it.

Finally, consider prioritizing the automation of tests that must be run within multiple hardware or software environments. Performing these tests manually tends to take longer and pose more challenges than manual testing within homogeneous environments, because testers need to be familiar with each type of environment in order to perform the tests manually. With test automation, once you have written the tests so that they work within all of your target environments, you'll be able to support a broad set of configurations with minimal effort.

Question 7: How Do You Integrate Automated Tests into Your Workflow?

Like all types of QA work, automated testing delivers the greatest value when it is integrated seamlessly within the rest of your software delivery pipeline. Tests that are run within a QA silo will undercut the overall efficiency of your team, even if you can automate the tests.

Ultimately, your goal should be to achieve continuous testing. Under a continuous testing model, automated tests are integrated into every stage of your CI/CD pipeline.

Achieving continuous testing requires several steps:

- Destroying the walls that silo your test engineers from the rest of your team. QA engineers should be plugged into all stages of the CI/CD process, from development to deployment.
- Rethinking the culture of testing. Not just QA engineers, but every member of your software delivery team should recognize the value of testing and strive to integrate it continuously into the delivery pipeline. In other words, DevOps teams should discard the notion that testing is the responsibility of QA engineers alone.
- Seamless communication across the various stages of the delivery pipeline in order to enable different teams to react quickly to the results of tests.
- Rethinking testing timelines. Traditionally, testing has been performed around the midpoint of the delivery pipeline: after code was written but before it was staged and prepared for release. This type of testing remains important, but forward-thinking organizations are adopting shift-left testing and testing in production.

When you can do these things, you achieve the goal of being able to test at every gateway of the software development life cycle.

Conclusion

The responses that your organization will take to address the automated testing challenges described above will vary according to your specific needs and goals. No matter which approach you take, however, you will benefit from having an automated testing partner like Sauce Labs.

Through a cloud-based test grid that supports thousands of software environment configurations and offers access to thousands of real devices for mobile testing, Sauce Labs makes it easy for organizations of any size to begin executing automated tests. The platform supports an extensive list of automated testing frameworks for native and browser-based applications on Mac, PC, iOS, and Android. With enterprise-grade security features available to all users, as well as advanced analytics and extended debugging functionality, Sauce Labs enables simple and secure automated testing from the convenience of one of the world's largest cloud-based test grids.

We invite you to sign up for a Sauce Labs free trial today.

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.

saucelabs.com/sign-up

FREE TRIAL

For more information, please visit
→ saucelabs.com

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

SauceLabs