WHITE PAPER

MAKING THE MOVE TO AUTOMATED TESTING

7 Key Questions and Answers

If you work in DevOps or software QA, you likely 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, and enables shift-left testing for earlier detection of bugs.

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

SAUCELABS

TABLE OF CONTENTS

3 Question 1: Which Testing Strategy Should We Use?

4 Question 2: Which Test Automation Framework Works Best?

5 Question 3: Which Test Grid Infrastructure Do I Need?

6 Question 4: How Much Should You Automate?

7 Question 5: Which Tests Should You Automate First?

7 Question 6: Which Tests Should Remain Manual?

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

9 Conclusion

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. These are just some of the uncertainties that make it difficult to put test automation into practice.

These challenges are understandable. Widespread automation for all kinds of software testing is a relatively new idea. Although the automation of integration testing was at the forefront of the DevOps movement, it took longer for organizations to recognize the value of other types of testing. DevOps teams remain unsure in some cases about how to integrate automated tests with the rest of their workflows, or which tools are available to help them do it.

These challenges to test automation may seem intimidating, but they are perfectly feasible to address. Toward that end, this whitepaper provides answers to the seven most common questions that software delivery teams must answer in order to adopt test automation successfully. It offers an overview of each problem and solution, along with links to external resources that discuss the challenges in greater depth.

QUESTION 1: WHICH TESTING STRATEGY SHOULD WE USE?

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

The first is unit testing. Unit tests, which are generally 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. A simple example of a unit test, and a discussion of the limits of unit testing, are available here.

A second common testing strategy 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.

The third and most sophisticated testing strategy 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.

Tests Are Not Either/Or

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 strategy, and one only.

In reality, these are not mutually exclusive testing strategies. Most organizations that perform a significant number of automated tests will use each of these strategies 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 choices for 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.

Of course, you do not have to limit yourself to one testing framework. It is 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 list here, but 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.

Appium

Appium, which was derived from Selenium, extends automated testing functionality for QA engineers to mobile apps, which Selenium does not directly support. Appium is likely the most popular automated testing framework for mobile, due to the fact that it supports both Android and macOS apps.

Selendroid

Selendroid is another Selenium-like testing framework designed for mobile apps. Formerly, 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 apps during development. It supports a broad range of tests for macOS environments. For more details on XCUITest, refer to the 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. Again, refer to the "Beyond Appium" whitepaper 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 apps 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.

Choosing a Framework

Again, you don't (and in most cases will not) rely on just one automated testing framework. Most organizations instead use multiple frameworks at once to cover different needs or use cases. By choosing a testing platform (such as Sauce Labs) that supports all of the leading frameworks, leveraging multiple frameworks at once is very easy.

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-premises and cloud-based. An on-premises 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 plan to include real mobile devices, of which there are currently tens of thousands of distinct models in existence. Even if you choose to include only a fraction of those mobile devices, acquiring and maintaining them is very 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, and it provides access to a vastly larger combination of software and hardware environments than most organizations could afford to manage and maintain in their own internal on-premises 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-premises grid is constricted by the size of the host infrastructure.

For a complete discussion of the respective advantages and disadvantages of on-premises 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 which 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 next section). Most organizations don't automate even half of their tests, although the number that do is climbing.

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, although what you choose to automate is just as important as how much you automate. As the following section explains, DevOps teams should identify which tests will deliver the most value when they are automated, and prioritize those for automation overhauls.

QUESTION 5: 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 6: 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 Sauce Labs contributor Ashley Hunsberger has written, the following types of testing are sometimes best performed manually:

- *Exploratory testing.* Automated testing is a great way to ensure that features work as you intend. If you are unsure which features your users

want in the first place, however, some manual exploratory testing can help you to identify them.

- *User experience testing.* In some cases, automated tools just can't fully capture the emotions, idiosyncrasies and other variables that shape overall user experience. For example, if you want to understand why users are abandoning your mobile app, manual testing that tracks the user journey and identifies when users cease using the app can prove insightful.
- *Accessibility testing.* Here again, automated tools cannot always track the issues that might arise when building effective accessibility features into an application.

This does not mean that these types of tests need to be performed manually in every case. Overall, however, testing that involves a subjective human element may yield more accurate results when performed manually than testing with a scope that is limited to whether software runs as expected or is compatible with a given configuration.

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, 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. *Every* member of your software delivery team (not just QA engineers) 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.

- Achieving seamless communication across the various stages of the delivery pipeline in order to enable different teams to react quickly to the results of tests. ChatOps can be useful for helping to achieve this goal.
- 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.

By adopting continuous testing, you make tests more efficient while also ensuring that tests deliver maximum value at every stage of the CI/CD pipeline.

CONCLUSION

How your organization addresses 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 over 800 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 PCs, 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 free Sauce Labs trial today.

ABOUT SAUCE LABS

Sauce Labs ensures the world's leading apps and websites work flawlessly on every browser, OS and device. Its award-winning Continuous Testing Cloud provides development and quality teams with instant access to the test coverage, scalability, and analytics they need to deliver a flawless digital experience. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP and Adams Street Partners. For more information, please visit saucelabs.com.

SAUCELABS

SAUCE LABS INC. - HQ
116 NEW MONTGOMERY STREET, 3RD FL
SAN FRANCISCO, CA 94105 USA

SAUCE LABS EUROPE GMBH
NEUENDORFSTR. 18B
16761 HENNIGSDORF GERMANY

SAUCE LABS INC. - CANADA
134 ABBOTT ST #501
VANCOUVER, BC V6B 2K4 CANADA