



# Guide to Mobile Test Automation with Appium, XCUITest, and Espresso

## TABLE OF CONTENTS

3	Executive Summary	9	Who Will Run the Tests?
3	The Complex Mobile Development Landscape	9	Is the App Cross-Platform?
4	Leveraging Mobile Test Automation	10	Engineer Background and Skill Sets
5	Mobile Test Automation Options	10	Is Full Source Code Available?
5	Choosing the Right Mobile Test Automation Framework	10	How Fast Does the Development Pipeline Move?
6	Understanding Cross-Platform vs. Native Frameworks	11	Comparison Summary
6	Advantages and Disadvantages of Appium	12	Leveraging a Test Cloud
7	Advantages and Disadvantages of Native Frameworks	13	Conclusion: How to Select a Mobile Test Automation Framework
9	Choosing the Right Framework: Appium vs. Espresso vs. XCUITest	13	Scaling Mobile Test Automation with Sauce Labs

## EXECUTIVE SUMMARY

The landscape surrounding mobile test automation is becoming increasingly complex for a variety of reasons. Not only are mobile device, operating system, and browser combinations growing more numerous all the time, but developers seeking to leverage mobile test automation must also decide between multiple frameworks to find the right testing solution for each app.

This is a challenging task given the many variables at play in choosing a testing framework. Teams must consider not only the technical dimensions of testing frameworks (such as which languages they use to write tests), but also the relative pros and cons of the different frameworks for supporting strategies like shift-left testing or enabling engineers in varying roles to write and execute tests.

To provide guidance on these questions, this whitepaper helps teams understand the key mobile test automation solutions available today and determine which ones to use for which testing needs. As the following pages explain, there is no single test automation framework for mobile apps that is best for all needs and use cases. Instead, the right test automation tool for a given job will depend on the nature of the application being tested, as well as the resources and priorities of the development team.

## THE COMPLEX MOBILE DEVELOPMENT LANDSCAPE

Today, building high-quality mobile applications that delight users is a tall order. According to the [official Android website](#), there are more than 24,000 distinct models of phones designed to run Android, each with different screen sizes, input features, battery life expectations, and so on. Combine that with the various generations and variations of Apple iPhones, and it's clear that mobile apps need to be able to run across a dizzying array of different device types and hardware profiles.

At the same time, there are dozens of different versions of the Android and iOS operating systems, adding more variables that developers must manage in order to ensure a smooth end-user experience across whichever device/OS combination their users happen to choose. Matters become more complicated still in the case of browser-based or hybrid mobile apps, whose performance can vary depending on which browser or Web framework they run on.

Ensuring that applications perform well across all of these mobile host environments can be tremendously challenging because individual devices, operating systems, or browsers may have configuration quirks that cause bugs or unexpected behavior within mobile applications that developers did not anticipate when writing the application code.

Nonetheless, managing all of these variables is a must for mobile developers, especially in a world where customers [quickly abandon apps that under-perform](#), and where a majority of users expect apps to [start in as little as four seconds and respond to input in under two seconds](#), no matter which platform hosts the apps.

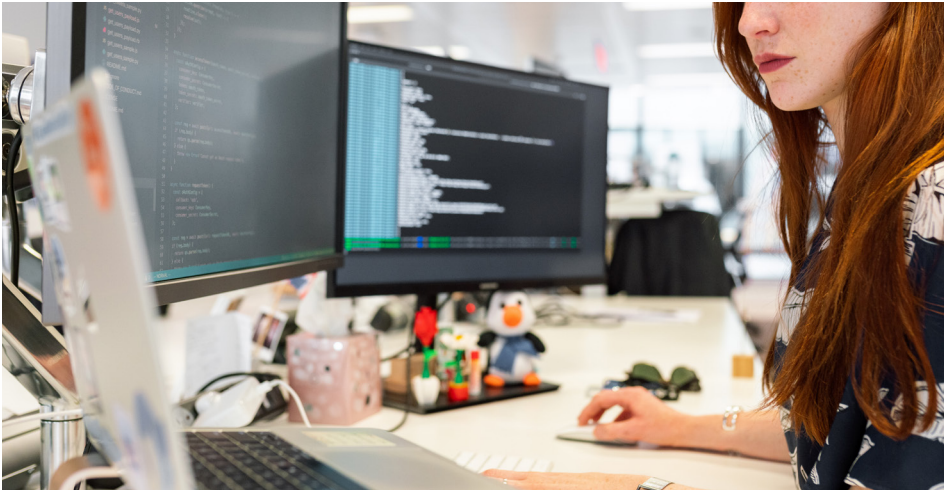
## LEVERAGING MOBILE TEST AUTOMATION

Given the mind-bogglingly complex set of variables that shape the software environments in which mobile apps run, how can mobile development teams ensure that their applications perform as required, no matter which mobile platforms their customers happen to use? The answer is mobile test automation.

Mobile test automation allows teams to write tests that automatically evaluate the performance of software during the development process, allowing developers to identify bugs before applications are deployed to end-users.

Without mobile automation testing, developers would have to test mobile applications manually by evaluating app performance by hand. Not only would such a strategy be time-consuming, but it would also allow developers to execute only a fraction of the tests that they could run when using automation.

With mobile test automation, however, teams can quickly and efficiently assess application performance and behavior across hundreds or thousands of different device/OS/browser combinations. They can catch and fix bugs that arise under certain configurations, then take steps to fix them so that all of their customers enjoy quality, high-performing applications. This is especially true when teams combine test automation with mobile device test clouds, which provide access to a wide variety of different mobile devices on which to run tests.



## MOBILE TEST AUTOMATION OPTIONS

Several mobile test automation frameworks exist to help development teams write and run tests. They fall into two main categories.

The first are cross-platform mobile test automation tools, which can work with any major mobile operating system and environment. Appium, an open source test automation framework first released in 2011, is the most popular option in this category.

The second category of mobile test automation solutions consists of native frameworks. These are testing frameworks that are built into specific operating systems. Espresso is the leading native test automation framework for Google's Android OS, while XCUITest is the main native option for Apple iOS testing.

## CHOOSING THE RIGHT MOBILE TEST AUTOMATION FRAMEWORK

With multiple mobile test automation frameworks out there, choosing the right solution for each mobile application or testing need can be a challenge. In the following pages, we compare the leading cross-platform and native test automation frameworks, explaining the pros and cons of each in terms of performance, test execution processes, and more.

The goal of this resource is to empower mobile development teams to choose the right test automation solution for each job and to ensure that they understand how to get the most out of the test automation routines they may already have in place (for example, by leveraging test clouds to provide simple access to as many device types as possible).

# UNDERSTANDING CROSS-PLATFORM VS. NATIVE FRAMEWORKS

As noted above, mobile test automation frameworks break down into two main categories:

- Cross-platform frameworks, where Appium is the leading solution.
- Native frameworks, with Espresso and XCUITest serving as the main solutions on Android and iOS, respectively.

Let's take a look at the pros and cons of each type of testing framework.

---

## ADVANTAGES AND DISADVANTAGES OF APPIUM

Appium offers several important advantages for many mobile testing needs:

- **Cross-platform support:** Perhaps most obvious, Appium is a cross-platform solution that developers can use to test almost any type of mobile application, no matter which OS it runs on or which language it is written in. (Indeed, Appium can be used for non-mobile testing, too, although other frameworks are more common for that situation.) This means that with Appium, teams can write a single set of tests and deploy them for both the Android and iOS versions of their applications.
- **Support for native, Web-based, and hybrid apps:** Along similar lines, Appium can test applications that run directly on mobile devices as native apps, as well as Web-based apps and hybrid apps that combine native and Web-based features. It's even possible to automate the operating system itself, which truly maximizes the flexibility of Appium testing.
- **Write tests in multiple languages:** Appium supports tests written in Java, Ruby, Python, PHP, JavaScript and C#. No matter which language your test engineers prefer, it's likely that Appium can handle tests written in it.
- **Support for testing complex interactions:** In addition to testing individual applications, Appium tests can be used to evaluate how apps interact with other apps, as well as with different browsers and operating systems. This makes Appium a highly flexible testing solution that can support a broad range of use cases.
- **Similarities to Selenium:** Appium is derived from Selenium, a popular test automation framework for non-mobile apps. For developers who are already familiar with Selenium, then, getting started with Appium is likely to be easier than having to learn a totally new framework.

These strengths make Appium a reliable choice for teams that need to test applications that will be deployed across multiple types of operating systems or browsers, and that want to do it all with a single test automation framework.

On the other hand, Appium has some drawbacks:

- **Slower test execution:** Compared to native testing frameworks, Appium tests take longer to run. This is due to Appium's testing architecture, in which tests live outside the app and are executed with HTTP calls. Slower test execution can slow down the overall development pipeline, especially if teams have a large number of tests to run and cannot run them all in parallel.
- **Testing code is separate from application code:** With Appium, teams typically write and maintain test automation scripts separately from application code. This practice adds complexity to the code management process and the software delivery pipeline.
- **The need for test customization:** Appium is a cross-platform framework that makes it possible, in theory, to write one series of tests that can run on both Android and iOS. However, in practice tests often need to be customized for each platform. When this happens, it becomes a drawback because it increases the work that test engineers must perform to run tests.



---

## ADVANTAGES AND DISADVANTAGES OF NATIVE FRAMEWORKS

Espresso and XCUITest come with their own set of pros and cons. Advantages of these native frameworks include:

- **Speed of test execution:** In most cases, native tests will run faster than those of Appium or another cross-platform framework. That is because native tests are not isolated from the application. They can also access application code directly, rather than having to use a method like HTTP calls to interact with applications.

- **Shared test language:** With native frameworks, tests are typically written in the same language as the application itself. This can make it easier to involve developers in testing if desired. It also makes it possible for tests to access internal application code directly, allowing engineers to create atomic and isolated tests that Appium would not be able to support.
- **Official platform support:** The native test frameworks are developed and officially supported by Google and Apple for their respective mobile platforms. That means that the native frameworks are kept up to speed with changes in OS libraries, features, and so on. As a third-party project, Appium can take time to catch up with platform changes.
- **Different tests for different platforms:** Because Espresso and XCUITest are tied to specific mobile platforms, the tests that engineers write using these frameworks will also be customized for those frameworks. In some respects, this can make it easier to write and maintain tests than in the case of Appium, where a single set of tests sometimes needs to be customized for each platform prior to test execution.
- **Test code lives alongside application code:** The tight coupling between native test frameworks and specific platforms also means that test code can be easily written and managed as part of the main application codebase, which simplifies the overall software delivery pipeline.

At the same time, the native frameworks are subject to some notable disadvantages:

- **No cross-platform support:** A key limitation is that native frameworks support only specific operating systems. That means engineers have to write and execute entirely separate sets of tests – one for each platform – if they are testing cross-platform apps.
- **Difficulty of testing user flows:** Compared to Appium, native frameworks allow little in the way of testing user flows by evaluating variables like application-browser interaction, testing how applications behave based on different device settings or assessing interactions between different applications.
- **Steeper learning curves:** Each native platform has its own set of tools and test execution routines. Engineers therefore have to learn each framework separately, a process that can take time and may require greater expertise on the part of the development team.
- **Focus on UI testing:** Espresso and XCUITest are primarily designed for UI testing, which means that application interfaces appear and behave as expected.



## CHOOSING THE RIGHT FRAMEWORK: APPIUM VS. ESPRESSO VS. XCUITEST

It would be a mistake to deem one testing framework superior to all of the others. Each solution is well suited for some use cases and poorly suited for others.

To evaluate which mobile test automation framework to use for which job, then, teams should ask themselves a series of questions.

---

### WHO WILL RUN THE TESTS?

Generally speaking, Appium is a testing solution designed first and foremost for Quality Assurance (QA) and test engineers. In contrast, the native frameworks cater to developers, who can use these frameworks to write test code alongside application code.

For that reason, organizations with dedicated QA or testing teams are more likely to find Appium to be a good fit, while teams without testing specialists may wish to stick with a native framework.

In addition, because native frameworks empower developers to write tests, they can be a useful means of enabling “shift-left” testing, which means running tests early in the development cycle. Instead of waiting for application code to be fully integrated, written, and built before running tests, developers can use native frameworks to test code before they push it into the pre-deployment environment.

The native frameworks are also more useful for teams aiming to perform “gray-box” tests, meaning tests in which engineers already have some understanding of the internal workings of the application and plan to use that knowledge to contextualize their interpretation of tests by writing atomic or isolated tests, for instance. In contrast, Appium is a black-box testing solution in which test data does not align with, or require special knowledge of, internal application architecture or code.

---

### IS THE APP CROSS-PLATFORM?

The most basic question to ask is whether the application you are testing is cross-platform. If the answer is no – if, in other words, it is an app that will run only on Android or only on iOS – it will likely be easier to write more sophisticated tests using the native testing framework of the target platform. That said, it could still make sense to use Appium for a single-platform app if engineers are already familiar with Appium (or Selenium-based) test frameworks.

If the app is cross-platform, it may or may not make sense to test it using a cross-platform framework like Appium or a combination of native frameworks. The answer in this case will depend on the other factors outlined below.

---

## ENGINEER BACKGROUND AND SKILL SETS

Along similar lines, it's important to consider the skill sets of the teams that will be writing and executing tests. If your engineers are already familiar with Selenium, then Appium may be a better mobile test automation choice than a native framework because it will require less time for the team to learn.

On the other hand, if your tests will be written by developers, they may find it easier to write tests in a native framework – especially if they already know the programming languages that the native frameworks use to write tests (which are Objective-C/Swift in the case of XCUITest, and Java or Kotlin for Appium).

---

## IS FULL SOURCE CODE AVAILABLE?

The native testing frameworks require access to source code, while Appium does not.

In most cases, this difference is not important because teams that develop in-house mobile applications typically have full access to the application source code. However, in certain situations – such as an application that depends on upstream components whose source code is not available to the development team – it may only be possible to use a framework like Appium that doesn't require full source code.




---

## HOW FAST DOES THE DEVELOPMENT PIPELINE MOVE?

The fact that native frameworks enable faster test execution is an advantage for development pipelines where teams build new features rapidly – such as on a daily basis. In those contexts, waiting for tests to execute can slow the overall delivery pipeline and undercut the ability of the team to push out application updates quickly.

On the other hand, if developers are releasing new features only once every week (or longer), the ability to execute tests as quickly as possible is less likely to be a priority.

## COMPARISON SUMMARY

			
Application Type	Mobile Web, Native/ Hybrid Mobile Apps	Native/Hybrid Mobile Apps	Native/Hybrid Mobile Apps
Operating System	Cross-platform	iOS	Android
Languages	Language Agnostic	Objective-C/ Swift	Java/Kotlin
Key Users	QA, Test Engineers	Developers (iOS)	Developers (Android)
Enables	Shift-right	Shift-left	Shift-left
Testing Type	Black box	Gray box	Gray box
Source Code Required	No	Yes	Yes
Flakiness of Tests	High	Low	Low
Speed	Slower	Faster	Faster
Updates	Trails iOS/ XCode, Android/ UI automator changes	Current/ Up-to-date	Current/ Up-to-date

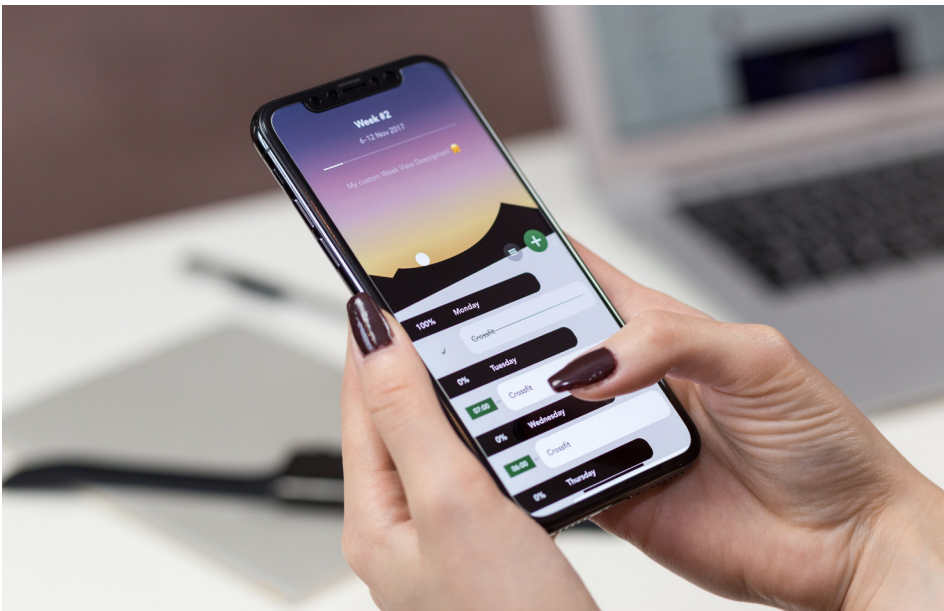
## LEVERAGING A TEST CLOUD

No matter which type of testing framework you choose, you can optimize the speed by running tests in parallel, while also improving test reliability, by running them in a test cloud.

Test clouds provide access to thousands of real or simulated mobile devices, making it easy to test applications across a range of different device profiles at once. Achieving the same device coverage using local testing infrastructure would be very complicated and expensive, because teams would need to set up and manage large-scale device emulators and/or physical devices, which would be a complex task.

Test clouds also offer the advantage of testing that can scale rapidly. Whether you need to run just a few tests or hundreds, a test cloud provides access to all of the infrastructure necessary to run the tests quickly, without the risk of delays caused by the exhaustion of available infrastructure.

And because test clouds from vendors like Sauce Labs offer enriched test reports that provide additional context about test results, they make it easier for teams to interpret test data and debug applications, regardless of which test framework or frameworks the teams use.



## CONCLUSION: HOW TO SELECT A MOBILE TEST AUTOMATION FRAMEWORK

Modern developers and test engineers have a variety of mobile test automation solutions available to them. While this level of choice is advantageous in that it makes it easy to select different frameworks depending on the nature of the application being tested and the priorities of the development team, the task of choosing the right framework can also present a challenge.

As your team evaluates the various options that are available, consider factors such as how many platforms your application needs to support, how rapidly you need the tests to execute, which members of your team are performing the testing, and whether techniques like shift-left testing are important to your team. The best testing framework for each application will vary based on factors like these.

Remember, too, that there is no reason why you can't use multiple frameworks at once. Not only is it common to use multiple native frameworks side-by-side for testing cross-platform applications, but it's also possible to use Appium and native frameworks at the same time, provided that your team has the skills to write and execute tests in each framework, and that it is manageable to maintain multiple sets of test code.

Whatever framework you use, however, don't make the mistake of allowing your mobile test infrastructure to constrict your ability to run tests as quickly and as broadly as you require. Instead, leverage a platform like the Sauce Labs testing cloud, which makes it simple to execute tests across hundreds or even thousands of devices at once.

---

### SCALING MOBILE TEST AUTOMATION WITH SAUCE LABS

Sauce Labs [mobile testing](#) solution supports popular mobile testing frameworks like Appium, Espresso, and XCUITest on a single platform allowing mobile app developers and QA engineers to maximize productivity by running automated tests on the framework of their choice, and working in the environment and language they already know and use. Sauce Labs uses a framework agnostic test orchestrator [saucectl](#) to execute Espresso and XCUITest tests (along with other frameworks) and enables teams to compare the test results across different environments and frameworks all in one view, thereby accelerating feedback loops and improving visibility into the application quality.



## ABOUT SAUCE LABS

Sauce Labs is the leading provider of continuous testing solutions that deliver digital confidence. The Sauce Labs Continuous Testing Cloud delivers a 360-degree view of a customer's application experience, ensuring that web and mobile applications look, function, and perform exactly as they should on every browser, OS, and device, every single time. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP, Adams Street Partners and Riverwood Capital. For more information, please visit [saucelabs.com](https://saucelabs.com).



[saucelabs.com/signup/trial](https://saucelabs.com/signup/trial)

**FREE TRIAL**