

# Solving the Biggest Challenges in the Mobile App Development Lifecycle


# Introduction

In today's fragmented mobile environment, it's no easy feat to develop mobile applications that are compatible with multiple platforms or to validate dynamic real-user conditions. Mobile QA teams face unique challenges when trying to scale mobile test automation and get complete visibility into an application's performance. While mobile app development teams struggle with gathering high-fidelity error & crash data and real-user feedback they need to iterate, debug, and release better mobile app versions faster.

Further, as the complexity and scale of technology, devices, and users has exploded, the sheer volume of errors has jumped dramatically. This has led to increases in the time it takes to capture, prioritize, and resolve errors. With users demanding the highest-quality app experiences, software teams can't afford slow detection and resolution times for the crashes or exceptions that impact their users.

User ratings and rankings in app stores also influence the lifecycle of an app and can contribute to its failure or success. Therefore, it's crucial to test mobile applications thoroughly using different scenarios to ensure compatibility and seamless functionality.

This paper discusses the biggest challenges modern mobile development and testing teams face throughout the mobile app software development lifecycle (SDLC) and provides practical solutions for overcoming them.



# Table of Contents

4	<b>Why is mobile testing important?</b>	8	Challenge #3: Mobile is Remote by Definition
5	<b>Part 1: Challenges in the Development and Integration Phase</b>	8	Challenge #4: Internal App Distribution is Complicated
5	Challenge #1: Mobile Ecosystem Fragmentation and Validating Apps for Dynamic Real-User Environments	9	Solution: Optimized Mobile Beta Testing and App Distribution
5	Solution: Cloud-Based Real Devices - No Maintenance and More Flexibility	10	<b>Part 3: Challenges in the Production Phase</b>
6	Challenge #2: Scaling Mobile Test Automation Early in the App Development Lifecycle	10	Challenge #5: Detecting and Prioritizing Crashes, Hangs, and Errors That Impact Users
6	Solution: Mobile Emulators and Simulators - for Testing Scale and Speed	10	Challenge #6: Long Resolution Times for Errors and Even Longer Times for Rolling Out Fixes
7	A Blended Approach to Cover the Breadth of Mobile Use Cases	11	Solution: Error & Crash Reporting in Production, and Across the SDLC
8	<b>Part 2: Challenges in the User Testing Phase (Alpha and Beta Testing)</b>	12	<b>Achieve Quality at Speed Throughout the Mobile App Development Lifecycle</b>

# Why is mobile testing important?

Mobile users expect rapid innovation – if your app isn't updated frequently, you risk bleeding users and losing your competitive edge.

Mobile users also have high expectations regarding the personalization and user experience of their apps. Users want information to be presented in context at the right time, in the right place, and in the right way. These demands and the tremendous opportunities they present bring mobile app development to the forefront as businesses compete for supremacy in today's mobile-first world.

Further, there are fundamental differences between the release cycles for mobile native applications versus web applications. It's much easier to develop and release updates for web applications that can be deployed at the same time for all users, multiple times per day. New versions of web applications can be accessed automatically by users and rolled back if issues are uncovered.

For mobile applications, release cycles are longer and more complex, which means fixing a bug for an app that's in production can be both costly and time-consuming. For example, an app installed on a user's device cannot be rolled back to a previous version—developers are forced to “fix forward” issues in future updates, potentially leading to multiple versions of the same app in production. Thus, testing earlier in the app release cycle becomes even more critical for native apps. In addition, since every app is installed separately, it's likely that different users will have different versions at any given time.

Without streamlined beta testing programs and error reporting systems, developers struggle to get real user feedback and detailed bug information in the pre and post-release stages. This information is critical to evaluating the overall app experience from the user's point of view and catching bugs before they impact users.

# Part 1: Challenges in the Development and Integration Phase

## Challenge #1: Mobile ecosystem fragmentation and validating apps for dynamic real-user environments

**Platforms:** iOS and Android operating systems comprise approximately [95%](#) of the mobile market share. The dominance of these platforms has transformed the mobile ecosystem, bringing application development to the forefront. And while a two-platform ecosystem may seem consolidated, going a level deeper into the many different operating system versions in play at one time reveals a high level of fragmentation, especially in the case of Android.

**Device Fragmentation:** Although there are fewer types of iOS devices than Android devices on the market, Apple continues to increase its range by introducing inexpensive and smaller versions of the iPhone and iPad to compete with low-end Android devices.

Android's open-source code further complicates the mobile ecosystem because manufacturers can change the way the operating system (OS) works on their devices. This means there can be tens of thousands of different Android versions in use at any given time. In addition, because hardware is custom made with different CPU, memory, and varying screen sizes, it is likely that a given app will behave differently on different devices. This makes it critical to test on as many different devices as possible.

**Application Types:** When creating a mobile app, developers need to decide which [type of app](#) is best suited for their business needs. Apps can be native, hybrid, or built over cross-platform frameworks such as Flutter, React Native, Xamarin and others. The decision to build a native app or use a cross-platform framework is usually based on architectural decisions, cost and speed of development, and the team's resources/skills. In cases where web components are already available or most developers are front-end developers, a hybrid app wrapping would make sense. And in the case where a new app is built from scratch and the team wants to have one code base for all platforms, Flutter or React Native may be a good choice. The application type may impact the way the app can be tested.

**Localization:** Each language supported by an app provides an opportunity to penetrate a new market and reach new customers. Developers and testers must consider that their apps will behave differently in different languages.

**Environment Considerations:** An app's behavior can be impacted by location, reception, battery, temperature, device accelerometer and more. Devices roam from network to network, cell tower to cell tower, making it important to validate apps for different networks and user mobility.

## Solution: Cloud-based real devices - no maintenance and more flexibility

Like real devices in a device lab, cloud-based real devices run tests on actual phone hardware and software. The key difference is that cloud-based devices are stored on a vendor's premises and are accessed remotely, which allows you to send test scripts to the devices over the Internet. These scripts are then executed on the devices and test results are sent back in the form of detailed logs, error reports, screenshots, and recorded video.

Maintaining real devices in-house is time- and resource-intensive and can pull focus from core testing activities. In-house device labs require a comprehensive range of devices to ensure confidence in the

quality of your testing efforts. You will also need to make sure the devices are updated and replaced regularly as new models come on the market.

However, a [cloud-based real device solution](#) gives testers instant access to the latest and greatest devices on the market. This eliminates the hassle of procuring and maintaining devices yourself while ensuring confidence in your device coverage.

Cloud-based real device platforms also provide better tools for monitoring and analysis. With an in-house device lab, troubleshooting is often done manually by a human running each test to replicate the error and find the root cause. With devices in the cloud, vendors can track and report on every step of the test and relay it back for analysis. This way a team member can simply look at a detailed error log, view screenshots, or watch parts of a recorded video to identify the cause. While this level of monitoring can be built into an in-house device lab, the effort takes months and requires adequate people resources that could be put to better use.

When choosing a cloud-based device vendor, it's important to consider criteria such as device access during peak times, breadth of device coverage, and security. Testing for real user conditions (network simulation, localization, GPS, etc.) makes a big difference in the quality of a mobile app.

A cloud-based solution can increase the flexibility, scalability, visibility, and cost-efficiency of your testing efforts, making it a key component of an effective mobile testing strategy.

## Challenge #2: Scaling mobile test automation early in the app development lifecycle

To validate features in development and catch common issues early in the SLDC, developers need the ability to run high volumes of functional tests (unit tests, smoke tests, and others) from their development environment. Teams should also run tests on every new commit pushed to a branch and ensure coverage by testing the application against a range of different mobile environments.

Both the choice of tools and methods used affect the quality of mobile app deployment and updates. Because software teams need to run tests on hundreds or thousands of environments in parallel, with each environment representing a specific configuration, it can be difficult to ensure both timely and cost-efficient testing if relying only on real device testing.

## Solution: Mobile emulators and simulators- for testing scale and speed

[Mobile emulators and simulators](#) are ideally suited to continuous integration (CI) pipelines because they're easy to provision and scale, providing you with comprehensive coverage.

Mobile emulators and simulators are also faster to provision than real devices because they are software-driven. Additionally, they enable parallel testing and test automation via external frameworks like [Appium](#), [Espresso](#), and [XCUITest](#).

Where Selenium revolutionized web app testing by pioneering browser-based test automation, Appium is its counterpart for mobile app testing. Appium uses the same WebDriver API that powers Selenium and enables the automation of native, hybrid, and mobile web apps. This delivers significant increases in testing speed for organizations coming from the manual world of testing on real devices. Similarly, enabling test automation with native frameworks (Espresso/ XCUITest) provides better test reliability, speed, and flexibility for native application testing.

Emulators and simulators enable parallel testing in a way that can't be achieved with devices in a lab. Because tests on emulators and simulators are software-defined, multiple tests can be run on tens of thousands of emulators and simulators at the click of a button without having to manually prepare each emulator/ simulator for the tests.

### A blended approach to cover the breadth of mobile use cases

A robust and complete mobile testing strategy requires organizations to have [a good mix of real devices and emulators and simulators](#). While emulators and simulators are complementary to real devices, they can't deliver the real-world environment that a device can. When used together in an automated testing environment, real devices and emulators enable modern mobile testing teams to get the most out of their mobile testing efforts. Plus, testing in parallel across multiple platforms helps to speed up tests while optimizing costs. With the right blend of real devices and mobile emulators/simulators, mobile teams can cover the entire breadth of mobile use cases.

KEY BENEFITS	Real Devices	EMU/SIM
Easy to scale and maintain	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Easy to provision	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Cost-efficient	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Detect Hardware Failures	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Advanced UI Testing	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Test Apps for Real-world Environments	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## Part 2: Challenges in the User Testing Phase (Alpha and Beta Testing)

We've covered some key challenges and solutions in the development stages of mobile app distribution. Now let's dive into the unique pain points for mobile teams when beta testing mobile apps.

The problem of **device fragmentation** is relevant even in the beta testing phases because it requires developers to test their app on as many devices as they can before releasing the app in the app store. This significantly increases testing time and development costs.

### Challenge #3: Mobile is remote by definition

Since mobile is remote by definition, developers cannot always see and understand problems that happen on a device that is in the hands of a tester or a user.

#### Limited user feedback makes remote debugging difficult for developers

The pandemic has greatly impacted the way we work and collaborate. Remote work in particular has presented unique challenges for mobile app development and testing teams.

In the pre-pandemic era, when a tester experienced an application issue or crash during testing, they could easily show the problem in real time, enabling developers to perform root cause analysis. Without physical proximity, it's challenging for testers to provide meaningful information to help developers understand and fix issues. Offshore QA teams experience similar challenges with respect to QA and developer communication, scheduling, and lack of control.

#### Reporting bugs on mobile devices is manual and time-consuming

In a remote environment, explaining what went wrong when an application does not work as expected requires developers to have screenshots, videos, logs, and other relevant information that will help them reproduce the issue. Users often don't understand how to report bugs, while for those who do, it is a manual and time-consuming process, ultimately leading to many bugs left un-reported.

### Challenge #4: Internal app distribution is complicated

Popular public app stores like Apple's App Store, Google Play, and the Samsung Galaxy Store each have unique requirements and policies for publishing an app. However, distributing apps for internal beta testing outside of the official app stores (also known as "dog-fooding") presents its own challenges. This use case typically involves the controlled distribution of mobile apps to internal company employees (authorized users) for beta testing, allowing them to test the app before it is released.

As software organizations deal with multiple app versions and a larger group of internal testers, manually reporting feedback can prevent beta testing participants from reporting bugs. Further, incomplete information and unclear reports on the bugs encountered, along with prolonged feedback loops between beta participants and developers, require development and testing teams to spend time filling in the gaps, causing delays in the timely public release of the app.



## Solution: Optimized mobile beta testing and app distribution

While there are several beta testing solutions on the market, the ability to streamline mobile app development and [beta testing processes](#) requires a cohesive solution that can empower development and QA teams to create consistent feedback loops throughout all phases of mobile development, including production, and further strengthens their manual and automated testing efforts.

Key benefits of using a developer-centric [beta testing solution](#) include:

### **Quick beta app deployment and real user feedback**

With the ability to leverage a SDK directly in their app and release beta app versions directly to the target users, developers can gather real-time real user feedback to create consistent feedback loops that provide way more depth and detail to further complement their scripted tests and better account for the edge cases.

### **Shorter development cycles, faster collaboration**

Solutions that integrate seamlessly with the entire ecosystem of tools can make beta testing painless and help cut down release cycles. Integrations with CI tools allow for apps to be automatically uploaded and get quickly distributed to the target users. Similarly, integration with bug tracking (e.g., JIRA, Trello) and team communication tools (e.g., Slack) allow teams to automatically file issues in the right places for the right stakeholders in remarkably less time.

### **Accelerated debugging and mobile app iterations**

Allowing beta testing participants to intuitively report bugs helps uncover more quality issues, and also provides meaningful information for developers to identify and fix bugs without spending endless hours analyzing bug reports. Further, the ability to view the actual user experience via video recordings can be valuable for developers and product teams to get insights into how real users use the app, and also reproduce issues faster by providing visibility into the events that happened before an app crashed or an issue was reported.

### **Better compliance and security**

Compliance with the major security standards and features such as data encryption, firewall, and audit logs can be especially valuable for setting up large beta testing programs. With secure and configurable app distribution teams can ensure that only authorized users have access to the beta apps and security is not compromised.

## Part 3: Challenges in the Production Phase

Software stability issues negatively impact user retention and a company's bottom line. Mobile app crashes and failures lead to fewer customers acquired and increased churn, which ultimately results in lost revenue.

Research studies from Intel, Harvard Business Review, and TechCrunch show that 44% of users report they would delete crashing apps immediately. 38% said they would do so if it froze for more than 30 seconds. For paid apps, 18% would still delete if the app froze for more than 5 seconds. Acquiring new customers can be up to 20x more expensive than simply retaining customers while increasing customer retention rates by just 5% could lead to an increase in profits by 25-95%.

Managing the stability of mobile apps in production is so difficult because the mobile app ecosystem is not a homogenous environment. The complexity around various operating systems, chipsets, form factors, memory, and performance criteria make it impossible to test for all scenarios. As a result, instability and other quality issues that were not encountered during the test and beta cycles will appear after an app is released. In the worst cases, these issues will negatively impact user retention and a company's bottom line, so it's important for teams to be informed and be able to solve these issues. Let's dive into the specific challenges.

### Challenge #5: Detecting and prioritizing crashes, hangs, and errors that impact users

When your development teams rely only on manual interaction and user reports to collect the data needed to diagnose application errors, there is a potential risk that crashes, exceptions, hangs, and low memory bugs can end-up impacting your mobile users.

The exploding complexity and scale of technology, devices, and users has caused a dramatic jump in errors. The time it takes to capture, prioritize, and resolve issues is steadily increasing. To sift through, understand, and investigate every kind of crash or exception from the high volumes of data (i.e., noise) coming from different sources becomes incredibly complex.

In a market where users have high expectations of software quality and a low tolerance for errors, development teams are always racing against time to fix issues that impact their users.

### Challenge #6: Long resolution times for errors and even longer times for rolling out fixes

Testing and error reporting is a team sport. When a bug is detected after release, consistent feedback and information sharing between technical support, QA, and engineering is often required to resolve the errors faster.

However, quick information sharing is impeded when the tools developers and QA teams use every day (Slack, Jira, GitHub, others) aren't integrated seamlessly. Further, the isolation of crash and error data from the team's workflows severely impacts their ability to [speed up resolution times for errors](#).

## Solution: Error & crash reporting in production, and across the SDLC

Crash and error reporting is not just an operational tool: it's a core part of how organizations mitigate application risk while protecting and growing revenue. The ability to get clearer signals from crash and error data has become critical for both developers and executives because high software quality directly impacts the user experience.

Using best-in-class [error and crash reporting solutions](#) not only helps you quickly observe and remediate errors in production, but also helps leverage those insights during the development and integration phases of CI/CD to improve test coverage. With deduplication, symbolication, indexing, and the elimination of false positives, these solutions enable teams to quickly capture, prioritize, and resolve the errors in their apps or devices while in development, test, beta, or production.

The key benefits of using error and crash reporting for mobile apps include:

### **Reducing “time-to-detection” to minutes**

It's critical to detect errors before your users do. Automated error capture across Android and iOS apps help teams never miss crashes, hangs, or low memory events.

### **Making error prioritization painless**

Solutions providing a flexible deduplication engine and powerful indexing/query capabilities allow teams to better assess the impact of fatal and non-fatal errors so they can more easily prioritize issues to fix.

### **Drastically reducing the “time-to-resolution”**

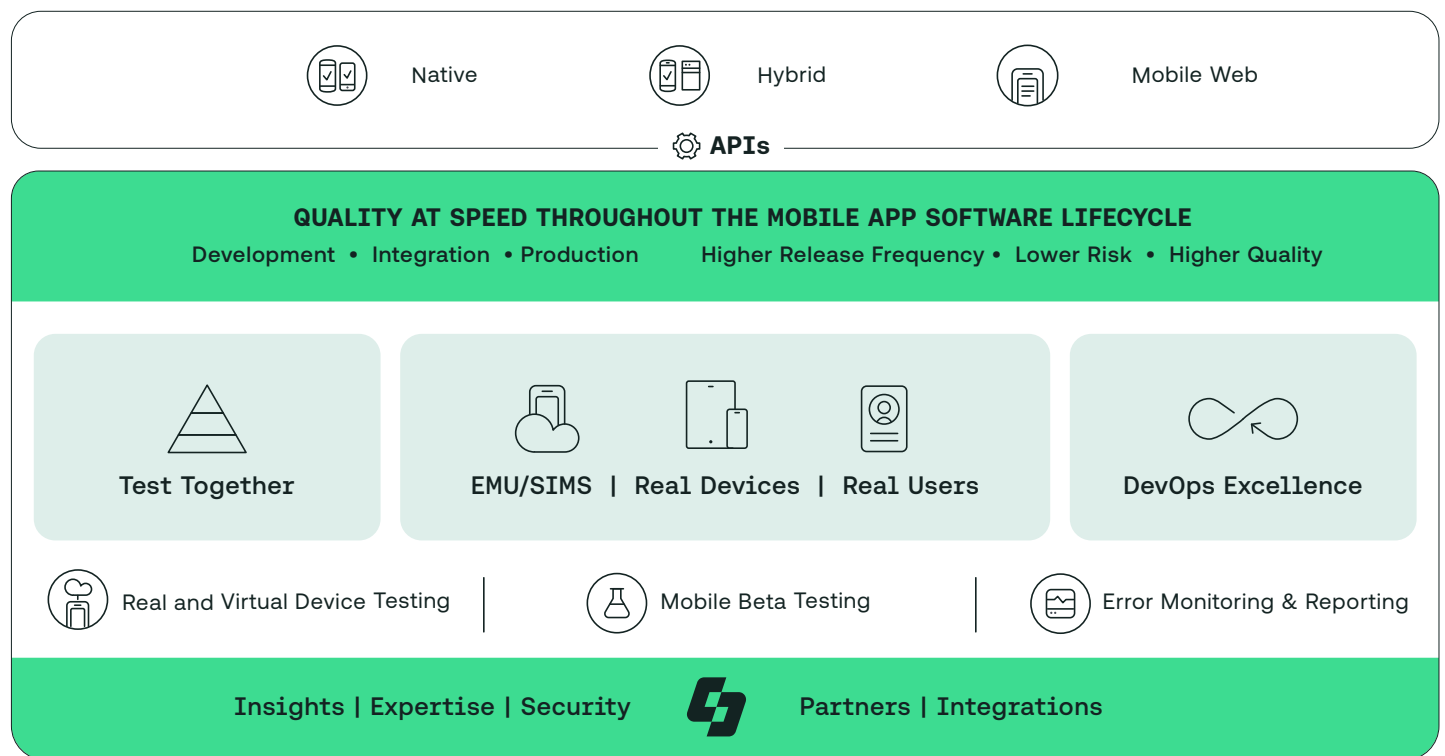
Deep integration with Jira, Slack, GitHub, DataDog, and other tools that teams use every day can help reduce time-to-resolution by hours, with the ability to create faster feedback loops, efficiently share error and crash information, and eliminate duplication of efforts.

# Achieve Quality at Speed Throughout the Mobile App Development Lifecycle

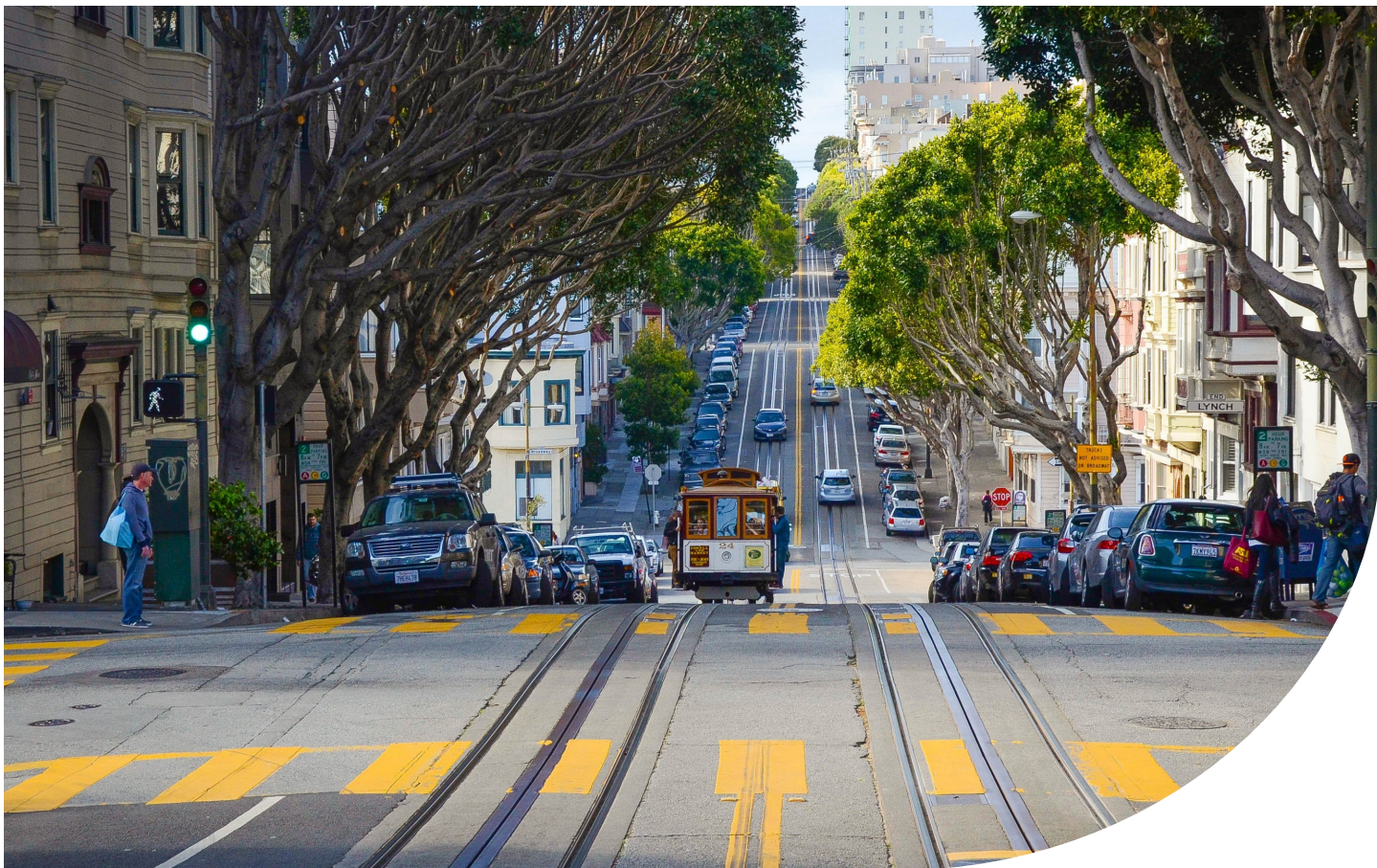
Delivering the best possible user experience and accelerating app releases, requires mobile development, QA, and extended teams to overcome unique challenges at each phase of the mobile app development life cycle.

To win in a highly competitive market and innovate with confidence, teams need end-to-end mobile quality solutions that can provide comprehensive device and test coverage, expedite beta app distribution and real user feedback cycles, and mitigate application risk throughout the SDLC.

Sauce Labs offers the leading [mobile test and error reporting solutions](#), enabling organizations developing mobile apps in the modern era of DevOps to deliver the best possible digital experiences to their users and achieve high quality, high-velocity releases. By providing an integrated test toolchain necessary to identify application risk and get enhanced visibility and insights across the entire software lifecycle, Sauce Labs helps your teams develop and release mobile apps with confidence.







## 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](https://saucelabs.com)



[saucelabs.com/sign-up](https://saucelabs.com/sign-up)

**FREE TRIAL**