Jump Start a Mobile Testing Lab

Contents

Initiating a testing program for enterprise mobile apps can sometimes appear to be an impenetrable wall. Mobility presents new challenges that, once understood, do have logical solutions, however. In several years of working with some of the world's largest enterprises, we at Mobile Labs have had the opportunity to observe some of the most successful mobile test lab programs.

The best practitioners manage to retain the traditional goals of QA testing while incorporating new requirements brought forth by mobile apps and solving new challenges mobility imposes on apps, devices, developers, testers, and users alike. Finding a point of entry means making decisions in eleven key areas, implementing plans based on those decisions, and then executing. In this paper we outline some of the decisions and programs we have seen in successful enterprises – the key characteristics of their testing programs.

Whether using a device plugged in to a desktop, a public cloud, or a secure, private device cloud, plan to test on real devices. The alternatives – software-based emulators – are widely recognized as appropriate only for basic checkout. Pre-mobility, it was relatively easy to conduct Web and desktop testing because both types of apps share their platform with the test tools. Mobility apps, however, target completely different computing platforms with different CPU, memory, graphics, and peripheral performance. Making the decision to use real devices for testing is the most basic step – plan for how the team will gain quick, easy access to real devices for development checkout, manual testing, and automated testing. Such a plan considers how to accommodate the needs of geographically dispersed testers – whether they are work-from-home types or a continent away.

Stephanie Rieger[1] notes that, "Testing on devices reveals all sorts of stuff that simply adjusting content never will," and she lists five key reasons to test on real devices: variations in device performance, capabilities, form factors, pixel density, and network impact(s). Kevin Kam of Qualitest[2] notes, "Since mobile applications are used on real handsets and not emulators, clearly the closer you get to the actual platform during the QA process, the better quality you will achieve."

A common roadblock for testers is iOS provisioning, the process by which Apple allows an app to be installed and launched on a real device. Properly provisioning an app (code signing with an appropriate certificate that belongs to a valid provisioning profile) requires membership in the Apple Developer Program. There are two levels of membership. An individual membership allows installing apps on up to 100 different phones or tablets in any given calendar year. An enterprise developer membership removes the requirement to register individual devices. In order for an Apple device to permit an app to install and launch, it must be signed with credentials issued by Apple to prove the app was created by a source known to Apple and that there has been no tampering with the app.

Other than by jailbreaking (a practice that most, if not all, enterprise customers have rejected), there is absolutely no way to avoid provisioning an iOS app before it will run on a real device. Successfully testing iOS apps therefore requires mastering the provisioning process.

The logistics that enterprises use for this process are varied. Some use a single enterprise wildcard provisioning profile and ask developers to sign apps with Xcode. Other customers use individual app profiles, one per app.

Whether developers are in-house or external, Apple[3] suggests using the enterprise's Apple developer account to perform app provisioning. Apple also suggests adding external developers to that membership rather than using credentials issued to the developer's employer, a move that keeps both companies in compliance with the Apple developer agreement.

If it is not possible for developers to sign and provision apps, some QA organizations obtain credentials under the company's enterprise membership and then sign or re-sign the apps before testing begins. Mobile app testing tools (manual or automated) that test using real devices should include a utility that can sign apps after they are built.

Device cloud management tools should be able to tell whether an app is provisioned properly and to provide appropriate diagnostic information if corrections are needed. Provisioning and code signing are initially difficult tasks. The processes are further complicated by geographically distributed test and development teams and outsourcing. Yet, Apple has made one thing certain: before the enterprise begins testing in earnest on real devices, it will join an Apple Developer program and will conquer the logistics of provisioning. Planning in advance how iOS apps will be provisioned represents a big head start.

Device cloud management tools should be able to tell whether an app is provisioned properly and to provide appropriate diagnostic information if corrections are needed.

Deciding to test on real mobile devices brings about the need to efficiently and securely share them. Small groups may be able to pass devices one-to-another using a physical checkout and paper tracking system, but most enterprises find a device cloud is a critical success factor when the number of testers gets larger than about a half-dozen or when testers aren't located near one another (separation can range anywhere from a different floor to another continent). A mobile device cloud is essential to prevent delays caused by devices that are in transit from one developer or tester to another or are checked out but not in use. The cloud should be secure and high performing, and should not rely on either Wi-Fi or cellular data links to carry test overhead traffic such as event notifications, screen shots, or object inventories. The latter is especially important when conducting network-based performance testing – a stream of test data in and out of the device's Wi-Fi connection can invalidate results.

Both manual testing and automated testing should be supported, with the ability to view the device in real time. It should be possible to manually manipulate the entire UI of the device so that settings can be changed when needed.

RapidValue[4], mobility and app development consultants, say it this way, "Testing with real devices can be expensive, depending on the way we explore it. It can also be disorganized and labor intensive [. . .] explore cloud-based testing tools if the requirement is to reduce project cost and [to achieve] high ROI."

Public mobile device clouds and private mobile device clouds have security implications. While arguing for the benefits of data and compute public clouds for many businesses, Jim Reavis, Executive Director of the Cloud Security Alliance, acknowledges there may be some difficulties for larger enterprises. Quoted by SearchCloudComputing[5], Reavis says, "There's this lack of visibility from the customer perspective -- businesses and enterprises -- not having control over the IT assets any longer. Not having visibility creates concerns about the unknown."

Jim Reavis
Executive Director
Cloud Security Alliance

While smaller businesses may gain security from public clouds, Reavis continued, "The bigger you are, the more sophisticated requirements you have. And you may have some challenges getting all of those [compliance and regulatory] requirements met when you don't have complete control over all the resources." A private mobile device cloud offers one exceptional benefit: apps and data remain under the enterprise's control, removing any reliance on external entities to protect them.

According to data compiled by mobile metrics vendor Crittercism, as of this writing there are 2,582 device types running 106 operating system versions serviced by 691 carriers worldwide. If the goal is to test every permutation possible, then every test case will be run 189,121,172 times! While public clouds often claim to solve this problem with "thousands of available devices," no one in the enterprise is likely to live long enough to achieve such comprehensive testing using a public cloud vendor who can swap devices once every 24 hours. And while the app is bottlenecked on testing, it is not in use by customers or employees.

Prem Phulara[6] of 360Logica points out, "Target devices for testing a mobile application should balance the need to use a representative sample of the expected device population with the need to optimize duration and cost of testing."

Luckily, the major mobile operating systems use logical screen sizes mapped to physical screens. Testing operating system versions on representative phones and tablets (throwing in a set for each CPU architecture) will get the needed coverage. Mixpanel[7] estimates that as of late April 2014, iOS7 runs on more than 90 per cent of iOS devices, iOS6 on eight per cent, and everything earlier comprises only 1.31 per cent. The situation with Android is not as striking, but Gingerbread runs on fewer than eight per cent of devices with the Android 4 versions running on 92 per cent of devices, the bulk of those on 4.1, 4.2, and 4.3.

ComputerWeekly.com[8] reported in a joint survey with TechTarget last year that nearly 50 per cent of US companies planned to allow their users to purchase their own smartphones and 30 per cent planned to allow users to buy their own tablets for use on the corporate network. Jennifer Lent9, who has explored the issue of more control over BYOD, notes, "Mobile experts have begun to question whether it's realistic for software teams writing and testing sophisticated mobile applications to support unrestricted company BYOD policies." So the news is bad: there are a lot of devices and a lot of operating systems. But the sheer numbers suggest needed convergence – the strategy must not be to test everything, but to test enough. It would be nice to return to the good old days of picking a single level of Windows Server, IIS, and SQL Server to support. Those days are gone, but there is no need to despair. Most enterprises test fewer than a dozen Android and maybe half that many iOS devices, tailoring their strategy to market numbers and adding additional devices as needed for additional operating system versions.

The task of sharing a well-constructed pool of mobile devices without a shared private mobile device cloud quickly becomes impossible. If a tester or developer's arsenal of devices is three dozen, there is no way to prevent testers becoming idle while waiting on devices. Buying one of everything for everybody might solve the problem, but a not-uncommonly-large pool of 100 testers and 50 developers could require 5,400 mobile devices, easily costing more than two million dollars at a cost of only $400 each. No device cloud means risking that developers or testers wait on a device to be relinquished, found, or worse – to be shipped and delivered.

Development, IT, and QA staffs have been cooperating for some time now to automate and field production environments. The various tools provided by Apple and Google to install apps on devices can be difficult to use, can be time consuming, or may require an administrator to manage.

Apple's and Android tools will install a properly signed app onto a USB-connected device, but locally-attached devices defeat the security and efficiency advantages of a private device cloud. Third parties have the means to move apps from cloud-based websites onto devices, but absent a full-blown MDM, such an install requires signing in to the website, using touch and other gestures – in effect, a manual, hand-operated process.

There may also be security concerns arising from storing the app in the cloud. One of the services a mobile device cloud must perform is enabling fast, painless movement of an app from the build system to the device, eliminating the need to use Apple or Google tools or cloud-based websites. Upon connecting to a device, a device cloud should be able to install and launch an app, and it should offer scripting capabilities that make it possible to automatically transfer new app versions to a database, onto one, more, or all devices, and to automatically run existing test cases against new app versions.

The device cloud should make it easy for both developers and testers to obtain real devices for checking out builds and fixes, and the cloud's automation capabilities should make it possible to set up production environments (correct device type, correct operating system version). At minimum, expect a device cloud to dramatically enhance efficiency in getting apps out of the build system and onto a real device for testing.

DON'T AUTOMATE EVERYTHING

Jean Ann Harrison[10], partner and senior technical consultant at Perfect Pitch Marketing, says that a sound mobile testing strategy uses manual testing where appropriate and automated testing where appropriate. She suggests that trainability tests, configuration tests, mobile device performance and usability testing are "faster, easier, and less expensive to run manually than to automate." Although it's possible to debate what makes sense to automate and what makes sense to do manually, it's a bet that some boundary condition testing and configuration based testing is easier and faster to perform manually.

Consider how the mobile device cloud supports efficient manual testing for what Harrison calls the "corner cases" and for exploratory testing. Instant, remote, real-time access and full gesture support are keys. Among other examples, Harrison cites tests involving operating temperatures and checks on app look and feel. Harrison suggests the best candidates for manual testing are easy for a human being to carry forward onto new devices or new operating systems but that might have both infrequent use and need major rewrites for new platforms.

FIND THE DEVELOPERS

Communication between the test team and the development team can shorten cycle times and is important for making sure apps are properly signed for installation on real devices. Screen sharing sessions and shared devices can make it quick and easy to demonstrate app performance problems, appearance, or logic problems over a live remote link, and if a device cloud provides a real-time view of the device, testers can easily record videos and share test run results with developers.

Diego Lo Giudice writes in ComputerWeekly.com[11], “Fast moving teams do not build code and then hand it off to a testing organization; they build code, deploy the application, execute it and immediately observe the results.”

To reduce the number of cases where an app has problems on a particular class of phone or tablet, share devices with the developers so code can be checked early in the process. A device cloud should make it easy to share a rare or rarely-used device so that developers can find and fix a problem. If the developers have access to devices during initial code development, they can more easily ship code that works the first time – so consider sharing the device cloud with them. Testers and developers should be able to collaborate instantly – even when separated by an ocean – using screen-sharing technology that lets the two re-create and understand mobile app issues. Problems can also be documented by desktop-based video recording; a movie can be worth a million words.

DON’T TRY TO FIND THE ‘SWISS ARMY KNIFE’ OF TESTING

Because enterprise mobile apps almost always use some form of back-end services, a thorough job of testing involves manual testing and automated UI testing, but may also involve security testing, service virtualization, load testing, performance profiling, and network condition testing. Most enterprises use tools that manage test cases – both automated and manual -- and that include versioning, storage of results, and the ability to data-drive and to automate concurrent testing. Theresa Lanowitz, founder of and analyst for industry analyst firm, voke, says, “It’s tempting to start evaluating tools looking for one that can perform all … functions – but several of these areas are specialties that have been in place for years and have best-of breed solutions already available, many of which were designed to test server back-ends for desktop Web applications.

Mobile apps use many of these same back-ends and so those tools may still be best of breed.” Choose vendors who are open to integrations among tool sets and that show leadership in their area of specialty. Consider, for example, how to factor mobility into load tests, service virtualization tests, or network degradation tests.

While a load simulator or network virtualization framework is exercising the back-end services, the question most asked in mobility is, “What is the performance seen by a typical user under current load conditions?” Look for tools that know how to combine traditional back-end testing with data sampled from interaction with a real device.

A UI automation framework that can measure transaction times and can detect when the user sees the expected result is the best way of answering this question and requires an automation script run on a real mobile device. Look for tools that know how to combine traditional back-end testing with data sampled from interaction with a real device.

Investigate the tools you have traditionally used for Web and desktop apps to see if the tools vendors have developed viable strategies for extending automation to real, cloud-based mobile devices.

A strategy based on new tools or languages may mean training costs and delays while everyone gets up to speed on a new way of doing business. Training, education and building infrastructure may represent significant hidden costs when implementing new tools.

Existing scripting tools are well suited for carrying out user interface testing if:

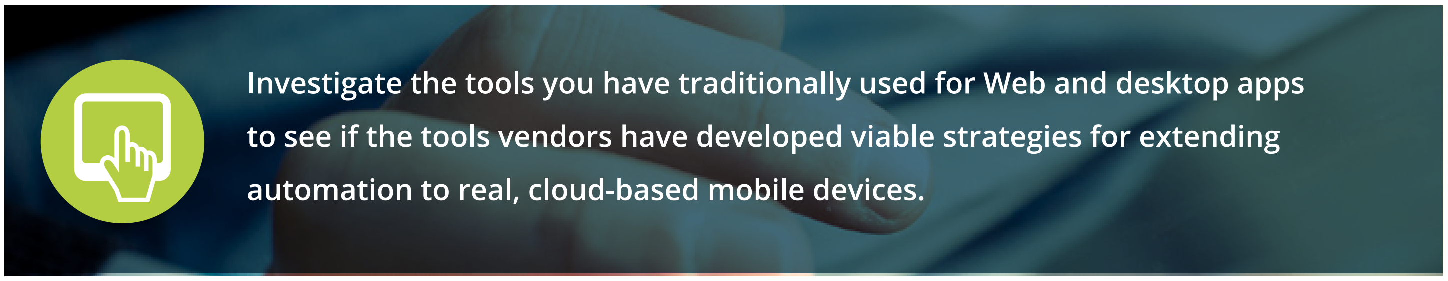
- They understand underlying mobile app object structures and are able to automate those objects using any of the three dominant app models (native apps, native apps with embedded web controls, and pure mobile websites).
- They can meet the above criteria while running an unmodified app on a secure, unadulterated, real mobile device.

Existing tools that meet these criteria represent a storehouse of expertise and infrastructure that can be quickly deployed to mobile app testing. If what you already know will get you a head start in mobile testing, go for it.

The key criteria should be:

1. **Do we already know how to use the tool?**
2. **Does the tool have stable support for real devices?**
3. **Can the tool solve my device access management challenges?**

These questions are especially important when evaluating open-source projects that are in mid-adoption cycle for mobile. For example, one open-source project recently deprecated its existing mobile device driver, replacing it with a new architecture for Android and a new driver for iOS – but the iOS driver did not support real devices at the time its predecessor was deprecated. Waiting for an open-source project to develop robust device support is likely to be rewarded with good results, once the community settles on a direction and works out the kinks. But how long can you wait? The open-source community's process prioritizes function and is rarely driven by schedules. If you need to test mobile apps now rather than later, you may want to favor existing, mature toolsets that have successfully transitioned to mobile and have a stable implementation – particularly if you can leverage the gains you get by using an existing base of skills and an existing ecosystem at the same time.



Investigate the tools you have traditionally used for Web and desktop apps to see if the tools vendors have developed viable strategies for extending automation to real, cloud-based mobile devices.

The greatest DevOps challenge occurs because mobility turns the client-server model on its head. Desktop Web tools tested back-end services in conjunction with a small set of browsers on a single desktop. Most important, however, the code for such browsers was almost always served up by a Web server, which made syncing the browser's function and the server's function pretty easy.

Mobile apps, however, install each build on a real mobile device – where it stays until replaced. Some care is needed to ensure against testing an old build. The process of quickly building production test environments is also challenging, requiring matching the right app version to the right platform (device and operating system level). The best defense is to use a mobile device cloud that can version apps and automate the process of moving apps from the build system onto the right devices without human intervention. Vendor tools from Apple and Google are difficult to use and make it easy to introduce errors into the process (wrong app, wrong device, wrong operating system version).

These difficulties have led to the popularity of Web-based app distribution, which makes it simple to manually download apps. But the process is still manually performed and can be error-prone.

The real solution is to automate the process so that apps are moved onto appropriate devices automatically by the build system.

Mobile devices are wireless and depend upon wireless network connectivity – public and private networks – for much of their function. So, testing must proceed not only on real devices, but also on real devices that have public network access. Mixing public network access by a mobile device with the ability to test the device from a system connected to your internal LAN is a source of potential concern for many CISOs. A device cloud must have a strategy to isolate both testing assets (apps, data, and network packets) from public access or exploitation, and must have a strategy for isolating a device's wireless access to the WAN from systems and data on internal LANS.

Moreover, public clouds often require copying apps and data across public networks to a third-party site where they must be unencrypted and installed. The public cloud by its nature cannot offer full control of these assets – or the devices – which may not be subject to your security controls and intrusion analytics. Use of devices tethered to individual tester desktops solves the public cloud exposure but may open a path into your internal network. The least secure solution of all is to connect to devices over their Wi-Fi TCP/IP stacks – this creates extra load on the wireless network, reduces performance of the testing solution and the app, and may require adding the mobile devices as peer computers on your LAN.

For most CISOs, the latter idea has proven to be the same kind of non-starter as jailbreaking. The best solution is to establish a test-only wireless network for WAN access that uses enterprise standard proxy and intrusion detection mechanisms while keeping that net separate and unreachable from internal systems. A private device cloud offering access to devices configured this way should use a restricted protocol like JSON over a specific port that can also be firewalled – a sort of "belt-and- suspenders" plan for the most secure setups.

Directing test data traffic (events, screen shots, object inventories) through the mobile device's USB data port helps isolate the wireless mobile devices from the internal LAN by restricting data flow to known ports and formatted messages. The resulting configuration can satisfy corporate security standards as well as satisfy the CISO that new exposures are not being created.

Sources

1 "Mobile is Just the Beginning", http://stephanierieger.com/on-designing-content-out-a-response-to-zeldman-and-others/

2 Kam, Kevin. "Mobile Emulators vs. Real Devices." QualitTest Group White Paper http://www.qualitestgroup.com/White-Paper-Mobile-Emulators-vs-Real-Devices

3 iOS Developer Enterprise Program, https://developer.apple.com/support/ios/enterprise.php

4 "Popular Online Mobile Application Testing Tools," http://www.rapidvaluesolutions.com/popular-cloud-based-mobile-application-testing-tools/

5 Boisvert, Michelle. "Real and Perceived Security Threats of Cloud Computing." Searchcloudcomputing.com N.p., May 2012. Web 15 May 2014.

6 Phulara, Prem. "Strategy of Mobile Testing." 360Logica Blog. http://www.360logica.com/blog/2014/04/strategy-mobile-testing.html

7 Mixpanel Report, https://mixpanel.com/trends/#report/ios_7

8 IT Priorities Survey, ComputerWeekly.com http://www.computerweekly.com/guides/IT-priorities-survey-2013%20-%20guideCategory2

9 Lent, Jennifer. "What is BYOD? Developers Redefine the Answer." TechTarget. N.p.,13 Feb 2013. Web 15 May 2014.

10 Denman, James A. "Mobile App Quality Takes More than Just Software Test Automation." Tech Target. N.p., May 2013. Web 16 May 2014.

11 Giudice, Diego Lo. "Why Agile Development Races Ahead of Traditional Testing." ComputerWeekly.com N.p., November 2013. Web 16 May 2014.