11 Mobile Application Testing

Computer technology changes rapidly. In a blink of an eye the computer went from the desktop to the laptop and now to the handheld mobile device. This migration has changed the way we conduct our lives, businesses, and governments. It has also significantly affected the way software developers and testers do their jobs.

Most software testing professionals find testing mobile applications very challenging—more so than almost any other software types or platforms. Actually, it's the devices and mobile environment more than the "application" that impose the challenge. These two components add many variables and complexities that may skew or mask problems in your application, which makes designing a robust test plan difficult. Briefly, you need to consider network performance and reliability, consistent user interfaces, transcoder influences, device diversity, and limited resource platforms.

In this chapter, we introduce a relatively new area of software testing: testing mobile and smartphone applications. We begin by describing the mobile application environment, which differs from that of a stand-alone application on desktops, laptops, and servers. Next, we enumerate the challenges of testing mobile applications—some of which we touched on earlier in this book. Finally, we cover some testing approaches and test-case considerations to help lower your learning curve in this new territory. After reading this chapter you should better understand the challenges and hurdles of testing mobile application.

Mobile Environment

With the widespread rollout of wireless hotspots, the line between mobile computing and "traditional" wireless network-based activities has blurred. Thus, to begin here we need to define the terms *mobile device* and *mobile applications*, with respect to the content of this chapter. In that light, we refer to a mobile device as one that has the capability to run network-based applications over a cellular or satellite data link. This encompasses most smartphones, tablets, and PDAs. That said, don't make the mistake of identifying mobile devices only by their appearance. Modern laptops can accept plug-in cellular or satellite cards, and some laptop devices have this access built in. Based on this definition of a mobile device then, a mobile application is a network-based program that runs on a mobile device.

This distinction is important. Yes, it is true that most mobile devices can use hotspots and wireless access points without a problem. However, those connections provide greater reliability and higher speeds than cellular networks, even with the adoption of 3G and 4G technologies. Thus, you design your mobile application with the expectation that it will use relatively slow, comparatively unreliable data links. You can also develop stand-alone applications, such as games, that run on a mobile device without the need to use the carrier's network. But for the purposes of this chapter, we do not consider stand-alone applications as mobile applications. Our focus is on the challenges associated with applications running on cellular data networks.

The key to creating successful test plans for your mobile applications is to understand the mobile computing environment. Table 11.1 identifies a number of crucial areas you should investigate while designing test plans. First, you must understand device connectivity issues and network speeds, regional availability, and latency. Keep in mind the underlying philosophy of this book: Your tests should not prove that your application works, but that your application does *not work* for the use cases. For example, if you have a location-based service or e-mail application, then your tests should identify software problems when the carriers network is slow or unavailable.

Next are three areas regarding devices—diversity, constraints, and input methods—which we cover in great detail later in the chapter. To create successful test plans, you and your testing staff must consider the numerous devices in the marketplace, the varying capabilities of each, and how the user interacts with the devices.

TABLE 11.1 Mobile Environment Test Design Considerations

Area	Comment
Connectivity	Device provisioning Network speed Network latency Network availability in remote areas Service reliability
Diversity Devices	Numerous web browsers to test Multiple versions of runtimes for Java or other languages
Device Donstraints	Limited memory or processor Small screen size Multiple operating systems Multitasking capabilities Data cache sizes
Input Devices	Touch screens Stylus Mouse Buttons Rollers
Installation and Maintenance	Installing and uninstalling Patching Upgrading

Last, you need to determine how you will install and maintain your application. Some vendors, such as Apple, maintain online stores where the user purchases the application, but only after Apple certifies your application for its platform. This makes installation and maintenance a little easier, as you have a single, certified distribution system.

Testing Challenges

As stated, mobile application testing is fraught with challenges. To help meet them, we can categorize most into four categories: device diversity, carrier network infrastructure, scripting, and usability. You need to think through each carefully when designing test cases. The shear combination of device types, operating systems, user input methods, and network concerns mean that trade-offs must be balanced with time, financial, and labor

resources to arrive at an economical test plan that detects most bugs in a reasonable time frame. Building a testing strategy that combines the methods discussed in earlier chapters will help.

In the rest of this section we discuss these categories and offer advice on how to tackle each one.

Mobile Device Diversity

The ever-expanding diversity of devices presents an often-underestimated and significant testing challenge to someone new to mobile application testing. It sometimes seems that manufacturers introduce new devices daily, making it almost impossible to keep up with the release cycles. Worse, more devices means more items to consider in your testing. Here's a simple example to illustrate only a few items you need to evaluate when a new device is released:

> Suppose Motorola develops a new method of text input via the touch screen for its Android-based phones. Can you design a test to determine whether the device's new input method breaks your application? If it does, can you fix your application without breaking support for other Android-based devices such as tablets? Can you even obtain a device to test? Do you have access to a supported carrier network?

Almost by definition, along with diversity of the devices comes diversity of operating systems, browsers, application runtime environments, screen resolutions, user interfaces, ergonomics, screen size, and more. You must be aware of all of these factors when creating tests. Device diversity also forces usability testing front and center, which at some point requires testers to evaluate your application on target devices. Using emulators is great way to start, but ultimately you will need to test real devices on real carrier networks.

This raises another facet of mobile application testing: testing on real devices versus emulators. From an economic standpoint you should do as much testing as you can with emulators. It may be financially unfeasible, even if you can obtain a device and access the wireless network, to test on the real platform. That said, emulators only emulate; they are not the real devices. So it is likely that you will observe differences between testing

with an emulator and the actual device. For example, the colors and shapes of buttons and input boxes may pass acceptance tests on an emulator but fail on the target device because of screen resolution and color depth differences between the device and a PC-based emulator.

In short, you need to realize there may be hundreds of mobile devices with the potential to access your application. Therefore, during the requirements-gathering and specification-writing phases, you will be called upon to make some tough decisions and choose a reasonable subset of devices to support and test. Be mindful that every device you do *not* test may not work with your application; hence, you may lose not only a customer but a customer base.

Carrier Network Infrastructure

Testing your application on a carrier network sets up another challenge. This is especially true if you want to support multiple carriers. Two of the highest hurdles to jump are: understanding and adapting to the carrier's infrastructure, and overcoming location-based obstacles.

Understanding a carrier's infrastructure is fundamental to developing a good test plan. Initially, you would think that your mobile application uses a carrier's network like an IP wireless hotspot. Not so. Figure 11.1 illustrates the "typical" infrastructure of most wireless carriers. The first difference is that the protocol is not IP-based; it is usually an RF-based protocol

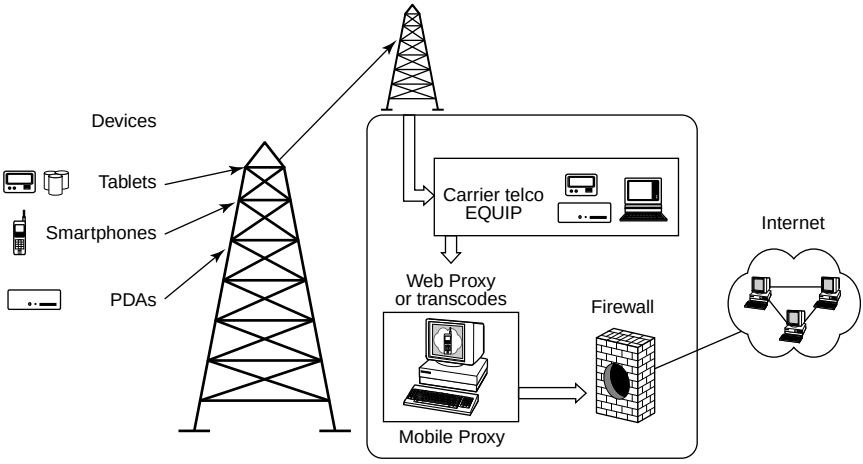

FIGURE 11.1 Generic Wireless Carrier Data Network.

such as code division multiple access (CDMA), time division multiple access (TDMA), or global system for mobile (GSM). The RF-based protocols treat the IP-based protocols as a "payload" and delivers them to the mobile device, which then decodes the payload and presents it to the application.

Also most carriers use some form of transcoder or Web proxy between the Internet and the device. These devices may perform a variety of functions. And, it is sometimes hard to determine exactly what occurs unless you work directly with the carriers. Often, they do not reveal this information for competitive purposes. The following is a short list of what may occur at a carrier's Web proxy or transcoder:

- Transform or transcode content into WAP or HTTP.
- Compress data for better throughput.
- Encrypt traffic for privacy and security.
- Block access to certain high-bandwidth sites.
- Strip HTML headers and other metadata from Web pages that your application may use.

Transcoding may cause UI inconsistencies across multiple devices. Some devices support Wireless Application Protocol (WAP) while others support HTTP. WAP uses Wireless Markup Language (WML) for content delivery. WAP and WML were intended to be the "standard" for wireless content delivery, but never gained a strong foothold. Nonetheless, numerous devices implement it, so you may encounter it during your tests. However, most smartphones and tablets support HTML and therefore rely on HTTP to deliver content. If you have UI problems across devices and carriers, check with each to determine whether WAP/WML or HTTP/HTML is being used.

Although data compression is intended to improve throughput, often during periods of high activity throughput may slow due to the overhead of compression. The same holds with security: Firewalls and similar layers may slow throughput during high-volume hours.

Finally, you must overcome location-based hurdles. Obviously, to test on a carrier's network, you need access to it. For instance, what if you have a travel application for a smartphone: How do you test carrier networks in other parts of the country or in other countries? Answer: You must travel there or hire someone there to test it for you. Both add to the cost of testing.

Scripting

An often-overlooked area of mobile applications testing is creating and running test scripts. Real devices do not allow you to load automated, repeatable scripts onto the device; test personnel manually execute all scripts. That is, someone walks through a written test script designed to find errors in a test case on the target device. Notice we said "target" device. There exist many targets in the mobile environment.

As we pointed out in previous chapters, manual testing is error prone. Unfortunately, it is unavoidable when testing mobile applications on real devices. As mentioned, most emulators have rich scripting functionality and can perform the bulk of regression testing and system tests. However, in the end you still need to have someone work with the device. (Later in the chapter, we explain how to create a generic manual test script to support multiple devices.)

The refreshing news is that mobile devices are becoming much more sophisticated and powerful. Given the competitiveness of the marketplace, it is reasonable to expect an automated scripting product to appear. Apple's iOS, Windows Mobile OS, and the Android OS are maturing rapidly, so it is likely that this problem may be a non-issue in future versions.

Usability

Usability testing presents challenges similar to those of test scripts. Recall from previous chapters that usability testing is mostly a white-box approach. Just like testing stand-alone desktop applications, a testing staff must manually try to find bugs in the user interface and user interaction layers of your application.

Unlike testing stand-alone desktop applications, mobile device testing involves more than one platform to test. For instance, you will want to search for UI consistency issues between Apple's products and the Android-based platforms. Although you are testing mobile applications, much of Chapter 7's discussions apply.

Testing Approaches

Some areas of testing mobile devices are similar to testing Internet applications, especially when evaluating the back-end infrastructures. The major

difference lies in how you approach testing the device itself. With Internet testing, you have only a handful of browsers to evaluate; with mobile devices, you have exponentially more.

Naturally, when testing back-end components, you should employ similar techniques and evaluate similar considerations as those discussed in Chapter 10, "Testing Internet Applications." Referring back to Figure 10.1, tiers 2 and 3 should have approximately the same configuration as a normal Internet application. As a quick review, you should test the performance specifications, data validation routines, and transaction processing components of tier 2. Testing tier 3 also is the same as with Internet applications; test response times, data integrity, fault tolerance, and recoverability on this tier. If possible test the tier 2 and 3 components separately from the device to ensure they meet your design specifications using function testing.

Testing tier 1, the user environment, differs from traditional Internet testing. The concepts presented on testing your content and website architecture still apply. However, user environment testing equates to device testing.

We should note the importance of use cases when developing test plans for your devices. Knowing who will use your application, and how and when, is imperative, as mobile applications have numerous points of failure. Table 11.2 lists items you might not generally consider when designing test cases for standard applications, whether stand-alone or Web-based. For example, testing your application on the carrier's network is extremely important. You want to find problems related to spotty coverage or sudden loss of connectivity. If your application involves data transfers, look for problems with data caching and incomplete synchronization with back-end data stores. What happens when coverage is suddenly restored after an interruption during an application download? Does a purchase occur twice? Check for bugs related to handling session reinitialization and data corruption. Some of these issues apply to Web-based applications running in a PC-based browser. However, LANs/WANs are much more stable. When dealing with cellular networks, you should expect to lose connectivity.

A test case specific to mobile testing is how your application handles incoming voice calls and text messages. Chances are end users will want to suspend your application, or run it in the background, while they answer the phone or read the text message. Try to build test cases where incoming calls and messages cause problems in your application.

TABLE 11.2 Test Categories for Mobile Application Testing

Test Category	Description
Install/Uninstall	Ensure the user can correctly install your application. Ensure the user can completely uninstall your application.
Network Infrastructure	Verify the application responds appropriately to loss of network. Verify the application responds appropriately to network restoration. Verify the application responds appropriately to weak signals.
Incoming Call/ Message Handling	Test whether user can accept calls/text messages while application is running. Test whether user can resume application when finishing calls/text messages. Test whether user can reject calls/text messages without disrupting application. Test whether user can initiate a call/text message without disrupting the application.
Low Memory	Ensure application remains stable when device encounters a low memory situation.
Key Mappings	Test that all key mapping works as specified.
Feedback	Ensure user feedback to keypress occurs within application design specifications.
Exiting	Verify that the application exits gracefully when initiated through pressing keys, closing the cover, or using the slider. Confirm the application meets design specifications when the user initiates a shutdown of device.
Charging	Ensure that application works as designed when entering charge mode. Ensure that application works as designed while in charge mode. Ensure that application works as designed when exiting charge mode.
Battery Conditions	Test how the application behaves on a low battery. Measure how quickly application drains the battery. Ensure the application responds per specification when the battery is removed while the device is powered on.
Device Interaction	Ensure the application does not overload the CPU. Ensure the application does not consume too much memory.

TABLE 11.3 Devices versus Emulators

Testing Approach	Disadvantages	Advantages
Real Devices	Expensive, especially if you target a broad base of mobile devices	Ability to test responsiveness of the application
	Inability to install metering or diagnostic development tools	Visual inspection of application on real device to verify UI consistency
	Unable to install on run test scripts	Test carriers' network responsiveness
	Network availability	Identify device-specific bugs
Emulators	Inability to identify device- related bugs	Cost-effective
	Underlying hardware may skew performance on real device	Easy to manage; multiple device support with single emulator

In the rest of the chapter we will cover some approaches to device testing in which you basically have two choices: test on real devices or use device emulators. Table 11.3 offers some advantages and disadvantages of each approach.

Testing with Real Devices

Manual testing with real devices is inevitable. Although costly, it has some advantages. Only by testing with the device can you experience its nuances and get a true feel for the user's experience. In addition, you can only test certain cases with real devices. Testing the reliability of a carrier's network and determining the effect of an incoming call or text message are obvious examples. On a real device you also can evaluate how your application behaves. Does it load fast and run at an acceptable speed? Does it look okay? Is the UI consistent across your target devices? Last but not least, you can determine device-specific bugs. This is almost impossible with an emulator. If you do find a device-specific bug, the challenge is to fix it without breaking compatibility with other devices.

Despite the advantages, testing with real devices also has some serious drawbacks. For example, it is costly because you must purchase the device,

as well as pay for carrier airtime. Neither is inexpensive, and if you are testing multiple devices from multiple carriers in multiple regions, the expenses grow accordingly. Some device manufacturers and service providers have devices that you can rent or access remotely, which may mitigate some costs. If you target an individual platform, such as the Apple iPhone family, you may be spared much of this expense. Still, you will need enough of each type (iPad, iPhone, iTouch) to test.

In addition, testing with real devices is a manual, white-box process. Someone must push the buttons, tap the screens, and enter data. As you know, manual testing is error prone, even with the best instructions and trained testers. Plus, it adds another expense to the process. You should keep exacting notes about each well-documented test script and its results. Then evaluate the effectiveness of the scripts and eliminate those with little or no value (i.e., fail to find bugs).

As we noted earlier, using real devices eliminates one important weapon in the software tester's arsenal: automated test scripts. Therefore, you should use written, manual scripts that specify generic actions, not details on how to perform the action on a device. Detailed test scripts for every device would be a challenge to create and maintain. In short order you would have a library of scripts, which may be obsolete when the device is updated. Generic scripts allow you to test system specifications across multiple devices.

For example, iPhones, iPads, and Android-based devices rely heavily on touch screens for user input. Other devices, such as BlackBerries or "standard" phones, have keyboards or keypads to allow for user input. Table 11.4 provides an example script to check whether your application, an e-reader, aborts if you receive a text message while reading an e-book. Notice the script does not specify exactly how to do any one step, only to perform the step using the user-input facilities of the device. These may be buttons, touch screens, or voice commands. At no point do you specify "Press OK" or "Press Send." This generic approach will allow you to evaluate test cases across multiple devices.

Last, manufacturers often "lock down" real devices, meaning you cannot load tools to monitor or debug your application. So when you hit a bug it is more challenging to isolate the problem. For instance, if your application is running slowly, you do not know whether it is the carrier's network, transcoding issues, your application, or a combination. Only by trial and error can you identify problems.

TABLE 11.4 Generic Device Test Script

1. Start e-reader application.
2. Open e-book.
3. Initiate SMS message to device from another device.
4. Verify SMS message alert is displayed.
5. Open SMS message.
6. Choose Reply to SMS message.
7. Compose SMS message.
8. Send SMS message.
9. Verify SMS message sent notification.
10. Return to e-book.
11. Verify e-book application is running.
12. Verify return to same page or bookmark.
13. Exit e-reader application.

Testing with Emulators

Testing with emulators may not be the preferred approach, but it is usually the most practical and cost-effective, and it even has some advantages. First, emulators allow for inexpensive and quick functional testing of your application. You can step through the application to find events and circumstances that do not meet the program requirements. Identify these bugs using emulators before you get into the expense of device testing.

Second, emulators are easy to manage, and because they run on PCs, every tester or developer can have an emulator. Developers can manage the software themselves, precluding the need of system administrators. Third, most emulator packages support multiple devices. To test a different device, just load a different device profile. Best of all, you incur no expensive carrier airtime costs. Fourth, emulators run on computers with more resources, such as faster CPUs and more memory. Fast response times during testing enables you to complete tests more quickly.

The last and probably most significant advantage is that most emulators employ high-level scripting languages, so you can create consistent, automated tests, which are less error prone and quicker than manual testing. Automated scripting also allows for easier and faster regression testing, which is especially important when verifying that changes made to your

application to support one device don't break support for another. The scripting languages in emulators generally are device-agnostic. Referring to Table 11.4, when you script Step 8, "Send SMS Message," the emulator will perform that function regardless of the device. This allows scripts to be used across devices.

The disadvantage of using emulators for testing is that you cannot identify the nuances and bugs of each device. As we've said before, at some point, you must test your application on the target devices. Without testing on real devices, you never can be 100 percent sure that you meet compatibility and performance specifications. Nonetheless, do not rule out using emulators for the bulk of your testing. It is a cost-effective and efficient way to eliminate most of your bugs.

Summary

Mobile application testing represents a new frontier in software testing. The mobile environment adds greater complexity and more interactions not experienced when testing standard stand-alone applications. That said, with an understanding of the challenges, you can greatly improve your chances of successfully testing your application.

Begin by trying to gain a handle on the device universe you want to support. Do you want to support only Android-based smartphones and tablets, or go for broke and support most major tablet and smartphone vendors? Next, understand the carrier's network infrastructure. Does it transcode, encrypt, compress, or in any way modify the data before sending it to the device?

You also need to find a balance between emulator and real device testing. Both have their pros and cons. Due to costs, you will likely use emulators more, and save device testing for the final phases. Use the test categories in Table 11.2 as a starting point to developing your own. Refer to the categories often when defining your test cases. Also, treat any written test script and result like source code; ensure you have adequate backups and some form of change control on the test documents. To save time and money, review the effectiveness of each script and eliminate ones that fail to add value.

Once you understand the fundamentals of mobile application testing, you should have no problems creating test plans and use cases. One thing is certain, mobile applications are here, and sooner or later you will need to learn how to test these unique applications. Why not start now?