21. https://code.google.com/p/android-test-kit/wiki/GoogleInstrumentationTestRunner

Espresso provides a small and easy-to-learn API to interact with the UI elements of a native Android app. Espresso is mainly aimed at developers who have access to the code base in order to write fast and reliable tests. However, if you are able to write Java code and have access to the code base of the app you want to test, Espresso is a nice little tool for writing the test automation. As an example, please refer to the code in Listing 5.7.

Listing 5.7 *Sample Code from Espresso*

Click here to view code image

```
onView(withId(R.id.login)).perform(click());
onView(withId(R.id.logout)).check(doesNotExist());
onView(withId(R.id.input)).perform(typeText("Hello"));
```

Espresso is able to execute tests either from the command line, from an IDE, or from a continuous integration server on real devices or emulators but not in parallel. However, test execution is much faster compared to that of any other Android test automation tools.

Additional useful information about Espresso can be found on the Google project page:

- Espresso start guide (https://code.google.com/p/android-test-kit/wiki/EspressoStartGuide)
- Espresso samples (https://code.google.com/p/android-test-kit/wiki/EspressoSamples)

More Android Testing Tools

As I said at the beginning of this chapter, the list of Android test automation tools mentioned in this book is by no means complete. There are so many open- and closed-source tools available on the market, and more tools are bound to follow. The following list contains the names of some other Android test automation tools that you should try. This list contains some closed-source enterprise tools as well as some unit testing tools.

- eggPlant (www.testplant.com/eggplant/)
- Experitest (http://experitest.com/)
- Jamo Solutions (www.jamosolutions.com/)

- Keynote (www.keynote.com/solutions/testing/mobile-testing)
- MonkeyTalk (www.cloudmonkeymobile.com/monkeytalk)
- Perfecto Mobile (www.perfectomobile.com/)
- Ranorex (www.ranorex.com/)
- Robolectric (http://robolectric.org/)
- Siesta (https://market.sencha.com/extensions/siesta)
- Silk Mobile (www.borland.com/products/silkmobile/)
- SOASTA (www.soasta.com/products/soasta-platform/)
- TenKod EZ TestApp (www.tenkod.com/ez-testapp/)
- TestObject (https://testobject.com/)
- UI Automator (http://developer.android.com/tools/help/uiautomator/index.html)

Android Tool Recommendation

Recommending a mobile test automation tool is not easy. There are so many factors that need to be considered when choosing a mobile test automation tool, and those factors are different for each app and project. From my point of view and judging from the apps I have worked with (social media and booking apps), I recommend taking a closer look at Robotium, Spoon, Appium, and Selendroid.

All of the tools are great to work with. They offer full support for native, hybrid, and Web-based apps. Besides that, all of the tools come with good documentation, code samples, and a great community if you want to ask questions. And last but not least, writing test code with the tools is very easy and lots of fun.

When choosing Robotium as a test automation tool, I highly recommend combining it with Spoon. Spoon’s test reporting feature is excellent, and the option to run your tests on several devices at the same time is unbeatable. Robotium is a well-developed Android test automation tool with a huge community and lots of support behind it.

Appium and Selendroid are also tools you should keep in mind. Both offer a great way to develop your automated tests in several programming languages. Both tools have great options for scaling your testing process in a cloud or a Selenium Grid.

Important

Keep one thing in mind: No matter which tool you use for test automation, use the resource IDs of the UI components if possible as doing so speeds up test automation and makes it more reliable.

iOS Tools

Let's go further with some iOS testing tools. What I already mentioned for the Android tools also applies to iOS tools:

- The selected tool list is not complete.
- I have included end-to-end test automation tools.
- All mentioned tools require coding skills.
- Before you start with iOS test automation, make sure you're familiar with the iOS UI structure of iOS apps.

UI Automation

UI Automation[22] is the iOS testing tool that is part of Instruments[23] provided by Apple. With the help of UI Automation, you are able to either record the tests or write them manually using JavaScript. If you are familiar with iOS apps, you know that iOS apps use so-called accessibility labels to describe the UI elements and make them accessible, such as for screen readers. Most iOS testing tools and UI Automation use these accessibility labels in order to communicate and interact with the app being tested. If your app has no defined accessibility labels, you are not able to write test automation for it.

22. https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/UsingtheAutomationInstrument/UsingtheAutomationInstrument.html

23. https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

UI Automation is able to simulate real user interactions such as tap, swipe, scroll, pinch, or type, either on a real device or on the iOS simulator. The code in Listing 5.8 shows some test actions that can be performed.

Listing 5.8 *Sample Code from UI Automation*

Click here to view code image

```
app.keyboard().typeString("Some text");
rootTable.cells()["List Entry 7"].tap();
alert.buttons()["Continue"].tap();
```

UI Automation is able to change the device orientation from portrait to landscape mode and back again. It is also able to handle different alerts that may occur during the test run on the mobile device. The automated tests can be executed from the command line, within the IDE, and from a continuous integration server.

More information about UI Automation can be found on Apple's developer pages:

- UI Automation JavaScript reference (https://developer.apple.com/library/ios/documentation/DeveloperTools/Reference/UIAutomationRef/_index.html)

Calabash for iOS

Calabash[24] is a cross-platform mobile test automation framework for native and hybrid Android and iOS apps (if you have already read about it in the Android section, you can skip ahead). The tool enables automated UI acceptance tests written in Cucumber.[25] With the help of Cucumber, you can express the behavior of the app you're testing using a natural language. This approach is called behavior-driven development (BDD), and it can be very helpful when business experts or nontechnical colleagues are involved in the acceptance criteria process.

24. http://calaba.sh/

25. http://cukes.info/

Important

I already described the features of Calabash in the Android tools section, so please refer back there. Almost exactly the same process is involved when writing the feature and step definition files for Android and iOS.

To get more information about Calabash for iOS, check out the Calabash iOS project site:

- Calabash for iOS (https://github.com/calabash/calabash-ios)

- Getting started with Calabash for iOS (https://github.com/calabash/calabash-ios/wiki/01-Getting-started-guide)
- Predefined steps for Calabash iOS (https://github.com/calabash/calabash-ios/wiki/02-Predefined-steps)

ios-driver

ios-driver[26] is able to automate native iOS, hybrid, and mobile Web apps using the Selenium WebDriver API. It uses the same approach as Selendroid but for iOS apps. It implements the JSON Wire Protocol in order to communicate and test iOS apps using Instruments. The tool is able to execute the tests either on a real device or on a simulator. Like Appium and Selendroid, ios-driver offers you a different set of programming languages with which you can write your test scripts. You can choose from the following:

26. http://ios-driver.github.io/ios-driver/

- C#
- Clojure
- Java
- JavaScript
- Objective-C
- Perl
- PHP
- Python
- Ruby

The code in Listing 5.9 shows some of the possible test commands for a native iOS app written in Java.

Listing 5.9 *Sample Code from ios-driver*

Click here to view code image

```
By button = By.id("Login");
WebElement loginButton = driver.findElement(button);
Assert.assertEquals(loginButton.getAttribute("name"), "Login");
loginButton.click();
```

In order to identify the UI elements of the app, ios-driver provides a UI inspector[27] similar to Selendroid that identifies and views the properties of the UI elements. ios-driver is able to handle localized apps and doesn't require any changes to the app under test. The tests can be executed from the command line or from a continuous integration server. Furthermore, the tool can be used as a node within a Selenium Grid to scale and parallelize the testing.

27. http://ios-driver.github.io/ios-driver/?page=inspector

More information about ios-driver can be found on the manufacturer's Web site, as well as on the GitHub project page:

- ios-driver getting started (http://ios-driver.github.io/ios-driver/?page=setup)
- Source code and samples (https://github.com/ios-driver/ios-driver)

Keep It Functional

Keep It Functional[28] (KIF) is an open-source iOS testing tool developed by the company Square.[29] KIF is able to automate native iOS apps using the accessibility labels provided by the app. This tool uses the so-called `tester` object to be able to simulate user inputs such as touch, swipe, scroll, and typing. Objective-C is used to write automated test scripts for KIF, and KIF is able to execute the tests on a real device or iOS simulator.

28. https://github.com/kif-framework/KIF

29. http://corner.squareup.com/2011/07/ios-integration-testing.html

Have a look at the sample code of Keep It Functional shown in Listing 5.10.

Listing 5.10 *Sample Code from Keep It Functional*

Click here to view code image

```
[tester enterText:@"user one" intoViewWithAccessibilityLabel:
@"User
Name"];
[tester enterText:@"Mypassword" intoViewWithAccessibilityLabel:
@"Login  Password"];
[tester tapViewWithAccessibilityLabel:@"Login"];
```

KIF can be fully integrated within Xcode to start and debug the test automation scripts. Furthermore, the automated tests can be executed from the command line or from a continuous integration server such as Bots.[30]

30. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/xcode_guide-continuous_integration/ConfigureBots/ConfigureBots.html

There's one thing you need to keep in mind when automating tests with KIF: it uses undocumented Apple APIs. This is not a problem when testing an app, but it's crucial that your test scripts not be part of the production code. If they are, Apple will reject your app due to the use of undocumented APIs. If you follow KIF's installation instructions, this should not be an issue.

Appium

Appium[31] is an open-source, cross-platform test automation tool for native, hybrid, and mobile Web apps (if you have already read about it in the Android section, you can skip ahead). Appium supports the mobile platforms Android, iOS, and FirefoxOS. Like Selendroid, Appium uses the WebDriver JSON Wire Protocol to drive and test the UI of the mobile apps.

31. http://appium.io/

Important

I already described the features of Appium in the Android tools section, so please refer back there.

More iOS Testing Tools

As I did for Android, I'd like to provide you with a list of additional iOS testing tools. The following list contains unit testing and end-to-end open- and closed-source testing tools; it is by no means complete:

- Experitest (http://experitest.com/)
- Frank (www.testingwithfrank.com/)
- GHUnit (https://github.com/gh-unit/gh-unit)
- Jamo Solutions (www.jamosolutions.com/)
- Keynote (www.keynote.com/solutions/testing/mobile-testing)
- Kiwi (https://github.com/kiwi-bdd/Kiwi)
- MonkeyTalk (www.cloudmonkeymobile.com/monkeytalk)

- OCMock (http://ocmock.org/)
- Perfecto Mobile (www.perfectomobile.com/)
- Ranorex (www.ranorex.com/)
- Silk Mobile (www.borland.com/products/silkmobile/)
- SOASTA (www.soasta.com/products/soasta-platform/)
- Specta (https://github.com/specta/specta)
- Subliminal (https://github.com/inkling/Subliminal)
- XCTest (https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/testing_with_xcode/testing_2_testing_basics/testing_2_testing_basics.html#//apple_ref/doc/uid/TP40014132-CH3-SW3)
- Zucchini (www.zucchiniframework.org/)

iOS Tool Recommendation

Recommending an iOS test automation tool is also not an easy task. Just as for Android, there are so many factors to consider when choosing an iOS test automation tool. I recommend that you take a closer look at ios-driver, Appium, and Keep It Functional.

All of the tools provide really good and powerful features in order to build reliable and robust test automation scripts for iOS apps. If you just want to automate a native iOS app, KIF would be a good choice as you can set up and write reliable and robust automated tests very quickly. Another advantage of KIF is that the test scripts are written with Objective-C, the same language with which the app will be written. If you struggle with Objective-C, you can simply ask your developers for support or have them write the test automation.

If you want to automate a hybrid iOS or Web app, you should use either ios-driver or Appium as both offer great support for various programming languages as well as the option to use them in a cloud or Selenium Grid environment. This provides powerful scaling and parallel test execution on several different devices and operating systems.

All three tools come with good documentation and very good code samples, are easy to use, and have a huge community behind them that is on hand to help if you run into any problems.

Mobile Test Automation Tools Summary

As you have seen, there are many different mobile test automation frameworks available on the market. Each tool has its own style of writing test scripts and supports different feature sets, different mobile platforms, and different mobile app types. Every tool currently available on the market has its pros and cons. You should really keep in mind that no tool is perfect, be it an open- or closed-source tool. Before selecting a mobile test automation tool, scan the market for possible tools and solutions to help you make the right decision. Use a sample app or a checklist to evaluate the various tools.

And last but not least, it's important to start simple with a mobile test automation tool. Don't try and find THE one and only test automation solution for your mobile app. Maybe you need to use more than one tool or to combine tools in order to build up a test automation suite that covers your needs and requirements. It is better to have only a certain amount of test automation in place that covers, for example, the critical parts of your app instead of every part. When choosing a tool, ask yourself the question "What should be automated?"

Continuous Integration System

Continuous integration (CI) is nothing new, and this development practice of integrating and testing the code from a centrally shared code repository several times a day has been in use for several years now. Every check-in is then verified by a different set of automated build steps to ensure that the latest code changes will not break the software and integration with other modules.

A CI server should be available in every project, no matter if the software is a desktop, Web, or mobile application, as it will help the team reduce the risk of broken software, give fast feedback to everyone involved in the project, and integrate smaller software parts into others as early as possible within the process.

Nowadays, there are plenty of open- and closed-source CI systems available on the market. If you have a CI system in your team, integrate the automated mobile tests into it. Nearly every mobile test automation tool can be integrated into a CI system. If this is not possible with the tool you're using, you'll have to find a way to integrate it such as with external build scripts that will run outside the CI environment to fulfill the task. This is very important for the project as a whole so that a complete build pipeline including all build and test scripts can be established.

When the test automation tool has been integrated, define a build and test strategy with your team. Talk to your developers and define which tests should be executed after every commit or during the night.

If your automated tests start to turn fast feedback from your CI system into slow feedback, split them into separate test suites. For example, you can define a smoke test suite containing tests that check whether the main parts of the application are still working. This test suite runs for only a couple of seconds or minutes and should be executed after every commit. Another test suite can be a regression test suite that runs, say, four times a day to check the app in more detail. And another suite can be a full test suite that runs every test every night to make sure the code changes from the previous day have not affected the existing parts of the app.

Another important point when adding a mobile test automation tool to a CI system is test reporting. The CI system must be able to display different kinds of test report formats in order to provide the whole team with visual feedback. The reporting component of the system should be easy to read and understand.

Once the CI system and all of the mobile testing and development tools have been integrated, define a complete build and test pipeline for your mobile application. The build pipeline should be able to start automatically without any user inputs, for example, by either listening to a central code repository or triggering the builds at a certain time during the night.

Furthermore, the build steps should trigger other build steps in order to unit-test, end-to-end-test, build an application on different staging systems, build alpha or beta versions of the app, or upload the application to a beta distribution server.

Here is an example of a possible, very simple mobile build pipeline:

1. Perform static code analysis, such as with PMD, FindBugs, Lint, or Checkstyle.
2. Perform unit tests.
3. Perform end-to-end UI tests.
4. Build a mobile app version on different staging systems.
5. Build a beta version of the mobile app.

6. Upload the beta version to a beta distribution system.
7. Sign and build a release candidate of the app (the only build step that should be triggered manually).

Build steps 1 and 2 should be executed on the developer's computer before he or she commits the code to the central repository.

If you have a CI system for your mobile app in place, don't forget to plug real devices into that server in order to execute all of the test automation on the real device.

The following are some of the available CI systems:

- Bamboo (www.atlassian.com/de/software/bamboo)
- Bots (https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/xcode_guide-continuous_integration/ConfigureBots/ConfigureBots.html)
- Buildbot (http://buildbot.net/)
- CruiseControl (http://cruisecontrol.sourceforge.net/)
- Janky (https://github.com/github/janky)
- Jenkins (http://jenkins-ci.org/)
- TeamCity (www.jetbrains.com/teamcity/)
- Travis CI (https://travis-ci.org/)

Important

Have a CI system in place and integrate your mobile test automation tool to get fast feedback about the quality of the app after every code change.

Beta Distribution Tools

As you have learned from the previous chapters, mobile users have expectations in terms of the usability, performance, and features of mobile apps. You and your team therefore have to be sure that your mobile app provides a great user experience and is fast, reliable, and fun to use. To meet all of these expectations, you and your team have a challenging job and need to test the app with other people to get feedback as early as possible in the development process.

To get this feedback from other users such as colleagues or users from your target customer group, you need a tool to distribute beta versions of your app. With the help of this tool, you can give potential users access to a beta version of the next release candidate.

Beta distribution tools include several useful features such as over-the-air app distribution, crash reporting, bug reporting, and direct in-app feedback. Some tools provide so-called checkpoints within the app where you can place questions for the user about the feature he or she just used. Another nice feature is so-called sessions, which can be included to track how the beta tester uses the app or a feature, thus helping you to identify unexpected app usage. The tools also provide data and statistics about the mobile operating system versions, device hardware, and interface language.

All of the information provided by a beta distribution tool is really important to know before your app is used by the majority of your target customer group. You can draw on this information to refine and develop your app in the right direction, thus making it far more reliable, stable, and of course fun to use.

When using a beta distribution tool, it is very important to inform the potential beta testers about all of these features and that information is gathered about the device and the user.

As a starting point, use a beta distribution tool within your company by asking your colleagues to test the app and provide feedback. Not every mobile app can be distributed as a beta version to the outside world due to network restrictions, company guidelines, or pertinent law.

Here is a list of beta distribution tools:

- Appaloosa (www.appaloosa-store.com/)
- AppBlade (https://appblade.com/)

- Applause SDK (www.applause.com/mobile-sdk)
- Beta by Crashlytics (http://try.crashlytics.com/beta/)
- BirdFlight (www.birdflightapp.com/)
- Google Play native App Beta Testing (https://play.google.com/apps/publish)
- HockeyApp (http://hockeyapp.net/)
- HockeyKit (http://hockeykit.net/)
- TestFlight (https://developer.apple.com/testflight/index.html)

Google and Apple also provide a way to distribute a beta app version to a wider user base. Within Google Play[32] you are able to add beta testers with their Gmail addresses who can then download the beta version from the Google Play store. Alternatively, you can define a staged rollout where a new version of your app is available only to a certain number of users, for example, 10% of the current user base. If the app works as expected, you can increase the app rollout either to your entire user base or by another increment.

32. https://support.google.com/googleplay/android-developer/answer/3131213?hl=en

On the Apple[33] side, you are also able to build a beta version of your app and distribute it to registered beta testers. The beta testers have to be registered with their unique device ID (UDID) using an ad hoc provisioning profile. However, you are able to register only 100 test devices within one membership year. You can get around this test device restriction if your company is part of Apple's enterprise program.

33. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html

Important

Wherever possible, use a beta distribution tool to gather early feedback from beta testers in order to build a better mobile app.

Summary

The fifth chapter of this book concentrated on mobile test automation. At the beginning of the chapter I explained the problem with the traditional test automation pyramid and mobile apps. I introduced and explained the flipped testing pyramid and showed a new pyramid—the mobile test pyramid. This pyramid contains automated, as well as manual, testing to fit all the requirements of the current state for mobile apps.

In the next section of this chapter I described the different approaches and types of mobile test automation tools. These are tools that use image recognition, coordinate-based recognition, text recognition, or native object recognition. I explained every approach, with pros and cons. Furthermore, I assigned the available tools on the market to the different approaches to provide you with an overview.

In another section I explained why it is a bad idea to just use capture and replay tools for your test automation. Those tools do provide a good starting point to get some test automation up and running in the first place, but in the long run they will cause lots of maintenance trouble and the tests aren't reliable at all.

In the section "What Should Be Automated?" I explained which parts of your app need test automation and which do not. For example, it is a good approach to automate the business-critical parts of the app. On the other hand, those parts that are likely to change often in the near future are fine for manual testing because the test automation will not run in a stable fashion if they are included.

To help you select the right test automation tool for your mobile app, I included a list with selection criteria to find the tool that fits best into your development and test environment.

The biggest part of this chapter covered the current state of mobile test automation tools for the iOS and Android platforms. I explained the different tools with code samples as well as pros and cons. I described the following tools:

- Robotium
- Spoon
- Selendroid
- Calabash for Android and iOS

- Appium
- Espresso
- UI Automation
- ios-driver
- Keep It Functional

The closing sections of this chapter covered the topics of continuous integration and beta distribution of mobile apps. I outlined a sample mobile CI build pipeline that can easily be adapted to your environment. I added a list of CI tools that can be used for mobile apps. In the beta distribution section, I explained the purpose of distributing a beta version of your app to colleagues or beta testers to get early feedback and bug reports.

Chapter 6. Additional Mobile Testing Methods

So far you've learned about mobile technologies and how to manually test mobile apps in different scenarios as well as while out and about. You've learned about mobile test automation and the concepts behind the different mobile test automation tools. You're now aware of several tools for the different mobile platforms and know how to select the right tool for your testing process.

To extend your knowledge and your toolbox, in this chapter I introduce some other possible mobile testing approaches: crowd and cloud testing. Both of these approaches can be beneficial in your daily work within a mobile team.

Crowd Testing

A company has three possible ways to establish software testing within an organization. Testing can be done with the aid of an in-house quality assurance department, via outsourcing (nearshoring/offshoring), or using a crowd testing approach. In-house testing and outsourcing are nothing new, and both are considered established approaches among various organizations and industries.

But this is not the case with crowd testing. The term *crowdsourcing* was introduced by Jee Howe[1] in 2006 and is a combination of the words *crowd* and *outsourcing*. In the software testing business the word *crowdsourcing* has become *crowd testing*.

1. www.crowdsourcing.com/

With the help of a community of external software testers, several crowd testing providers offer a new way to perform software testing. The external software testers come from diverse backgrounds, both geographically and in terms of their level of technical knowledge. Depending on the crowd testing provider, the crowd can range from a few people to several thousand testers worldwide, and it may include software testing experts and people of any age, gender, profession, and educational background. Furthermore, the crowd testers have lots of different devices with diverse hardware and software combinations and access to a number of different data networks. Crowd testing is tantamount to testing in the wild.

With the aid of crowd testers, an app will be tested under a set of realistic scenarios that can't be created by an in-house testing team. The mobile app will be tested under real-world conditions—with different data networks, hardware, and software as well as different users. External testers provide a fresh set of eyes for your mobile app and will doubtless report lots of bugs, deviations, as well as performance, usability, and functional issues.

Crowd testing providers offer a platform where crowd testers can register and create a profile stating their devices, skills, and demographic background. The client can add the app under test, the preconditions, sample scenarios, instructions, known bugs, and detailed test plans. The client is also able to define the demographic background, target customer group, skill set, and devices on which the crowd should test the mobile app.

Some crowd providers have a project management framework in place, including governance and legal structures for the testing phase. In addition, test providers assign a project manager who is responsible for the test cycle. The project manager is also the person who filters, rates, and categorizes the bugs reported by the crowd and summarizes the testing cycle for the client.

Some crowd testing providers have an assessment center or trial project in place where possible crowd testers must participate in order to verify their testing abilities before being accepted into the testing community.

Most crowd testing providers charge a fee for their services, but crowd testing is relatively cheap because you pay only for the package that you have agreed upon with the crowd testing provider. There are different kinds of packages available, such as simple bug reporting, exploratory testing, and/or executing defined test cases.

Figure 6.1 shows the typical crowd testing process from the first briefing with the crowd provider to the final presentation. During the test cycle, the client is always able to see the live progress of the crowd testers.

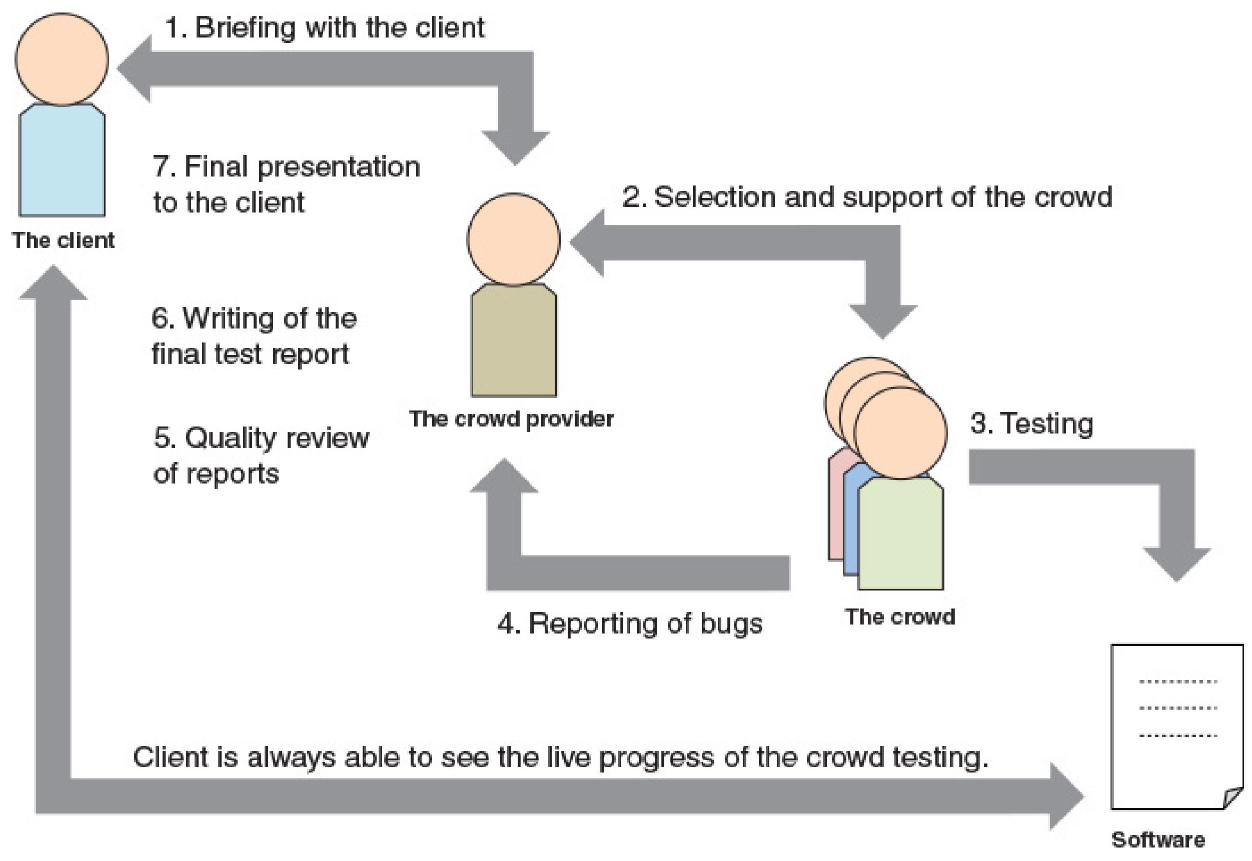

1. Briefing with the client
The client
7. Final presentation to the client
2. Selection and support of the crowd
6. Writing of the final test report
The crowd provider
3. Testing
5. Quality review of reports
4. Reporting of bugs
The crowd
Client is always able to see the live progress of the crowd testing.
Software

Figure 6.1 *Typical crowd testing process*

1. The first step is the initial briefing between the crowd provider and the client.

2. The crowd provider selects the crowd based on the client's requirements. During the test cycle the provider supports the crowd.

3. The crowd tests the software.

4. The crowd testers file reports depending on the aim of the test cycle, for example, bugs, feedback, or any other kind of problems.

5. The crowd provider ensures that the quality of the reports is good enough. He or she will follow up on bugs if more information is needed.

6. At the end of the test cycle the crowd provider writes the final test report.

7. The report is then presented to the client.

However, there are some challenges you need to be aware of if you want to use a crowd testing approach in your project.

It can take a long time to prepare and organize a crowd testing cycle. You need to define the exact goal for the test cycle and collect and prepare all of the information needed for the crowd in order to get valuable results. You need to brief the crowd testing provider, and at the end of the test cycle you'll need some more time to review all of the bug and test reports you receive. However, these steps are also necessary if you want to brief a new in-house tester on a new project.

The reported bugs may be of very low quality because of the crowd's lack of knowledge. The crowd testing provider will of course filter and categorize the bugs and pay testers only for those of real value, but it's also likely that the bug reports aren't detailed or precise enough for your needs.

It may be very difficult for crowd testers to access the development and test systems. Data privacy and security issues may prevent you from granting external access to internal staging systems, forcing the crowd testers to use the production environment which, in turn, has to work and interact with a beta version of your app. Maybe you need to create an isolated section within the production environment that can handle the beta requests and be used for the crowd testing cycle. At the end of the test cycle you need to be sure that the crowd testers are no longer able to access the production environment and that the app will be of no use to them.

Another challenge comes in the form of legal hurdles and NDAs (nondisclosure agreements). If your app is confidential, for example, crowd testing may simply not be an option.

The pros of crowd testing include the following:

- The crowd includes different testers from around the world, with different demographic backgrounds and skill sets.
- Lots of different mobile devices with different hardware and software combinations can be used for testing.
- The mobile app is tested in real-world conditions with real users.
- The crowd provides a fresh set of eyes for your application.
- Lots of issues will be reported.
- Crowd testing providers filter and categorize the bugs.

The cons of crowd testing include the following:

- Crowd testers are not generally testing experts.

- You don’t really know whom you’re dealing with.
- Bug reports may be of low quality.
- Access to staging systems can be very difficult due to legal hurdles, data privacy, and security concerns.
- Crowd testing can take a long time to prepare.
- Communication with the crowd can be difficult.
- Reproducing bugs can be difficult.
- There is a risk that the mobile app will continue to be used by the crowd after the test cycle has ended.

Important

Some crowd testing providers have a mechanism to automatically uninstall the app under test from testers’ mobile devices.

The following list includes some crowd testing providers (this list is by no means complete):

- 99tests (http://99tests.com/)
- Applause (www.applause.com/)
- crowdsourcedtesting (https://crowdsourcedtesting.com)
- Global App Testing (http://globalapptesting.com/)
- Mob4Hire (www.mob4hire.com/)
- passbrains (www.passbrains.com/)
- Testbirds (www.testbirds.de/)
- testCloud (www.testcloud.io/)
- TestPlus (www.testplus.at/)

Important

The crowd testing approach is a good extension of your in-house testing team. However, crowd testing won’t and shouldn’t replace your in-house testing activities.

Private Crowd Testing

If you're not able to use a public crowd testing provider because of legal restrictions, NDAs, or data privacy and data security concerns, or because you're not able to grant access to your development environment, you can use a private crowd testing approach.

You can start and build up an internal crowd testing session with your colleagues. Depending on the size of your company, you'll be surrounded by lots of people from different backgrounds working in various departments, so you can collect feedback from developers, designers, product managers, project managers, management, sales, and marketing colleagues. With the input from your colleagues, you'll be able to simulate real users in different usage scenarios to get an initial impression of your app.

One of the biggest advantages you'll notice when establishing a private crowd testing approach is that you'll need less time to prepare and organize it than with a public crowd testing approach because your colleagues are already familiar with the corporate environment, the product, and its features. Employees can also gain access to the development or staging environment, which allows you to sidestep any legal restrictions and NDAs.

Furthermore, it is very easy to communicate with your colleagues during the test cycle and to observe them while they are using the app. This will provide you and your team with extremely valuable insights into how users actually interact with your app in terms of usability and functionality. At the end of a test cycle you can interview your colleagues about the app and its new features to gather more information about their opinions of the app and any problems they encountered during testing.

The internal crowd testing session can be kept short and can be repeated several times during your app's development phase. This gives you more flexibility when it comes to reacting to possible usability, functional, or performance problems.

One way to keep your colleagues dedicated and motivated is to introduce your internal testing session as a competition. Try creating categories for the competition and award prizes for each category. Possible categories could include the following:

- Best usability bug
- Best functional bug

- Best performance bug
- Best security bug
- Best feedback provided
- Best bug report
- Best overall test engagement

Prizes could include the following:

- Company mugs
- Funny stickers
- T-shirts with funny slogans
- Vouchers

Prizes don't have to be expensive; they just need to motivate people to find and report as many bugs as possible. Furthermore, competition will give your colleagues an incentive to take part in upcoming test cycles.

Important

Try to create a private crowd testing session within your company and see how your colleagues perform as testers. You'll be surprised at the bugs and the reported results.

Mobile Cloud Testing

Manufacturers of mobile cloud testing solutions provide a wide range of current mobile devices with different hardware and software combinations in the cloud. Cloud testing providers use the characteristics of cloud computing to provide this service to mobile companies, teams, and mobile testers. Such characteristics include the following:

- The cloud uses a dynamic, shared, and virtual IT infrastructure.
- The cloud provides on-demand self-service.
- The cloud is scalable based on the load.
- The cloud is priced according to consumption.
- The cloud is available across different network connections.

Mobile cloud testing solutions are accessible via the Web and provide different types of testing that can be performed within the cloud on real or emulated/simulated devices. The different testing types available are:

- Functional testing
- Performance testing
- Load testing
- Mobile device testing
- Cross-browser testing

As is the case with Open Device Labs, the mobile cloud provides you and your team with easy access to a comprehensive range of current mobile devices using all of the different mobile platforms, no matter where you are in the world. If you want to test your mobile app in the cloud, all you need to do is allocate the physical device, upload and install your app, and start your manual or automated testing. Providers offer different kinds of additional services such as reporting features, screenshots, and videos of your test session or an API to execute your test automation scripts on several devices in parallel.

The fact that mobile test clouds are distributed in different geographical regions all over the world makes it easy to simulate different network technologies and scenarios from potential mobile users.

One major advantage of a mobile test cloud is that you don't need to buy new phones for your development or testing efforts as this will be done by the cloud provider. Furthermore, you don't need to maintain all the different devices, which may have a positive impact on your project costs.

However, there are also some limitations if you want to test your mobile app within a mobile cloud. For example, if your app uses a Bluetooth connection to connect to other physical elements such as speakers, it is not possible to test within a cloud. It is also not possible to use all the sensors and interfaces such as the proximity sensor, brightness sensor, acceleration sensor, or gyroscope sensor because the cloud devices are mounted and connected to a server within a data center. On top of that, you can't test your mobile app for interruptions or notifications from other apps or the device itself.

Another drawback of testing your app manually within a mobile cloud is that you're interacting with your app through the computer mouse. You have no physical contact with the device or app with your hand or fingers, making it very difficult to get a feel for the usability and response of the app. Furthermore, multitouch gestures can't be performed on the touch interface.

And, last but not least, security and privacy issues shouldn't be underestimated when using a mobile test cloud. You need to be sure that the provider will completely remove your app and its data from the test devices after the test session has been completed; otherwise the next cloud testing user might be able to see and use your app.

Before choosing a mobile cloud testing provider, check the provided features and compare them with those of other vendors. You should also weigh the pros and cons of mobile cloud testing to see if this approach fits your project and development environment.

The pros of mobile cloud testing include the following:

- Easy access to the physical devices
- Easy access to emulators and simulators
- Fast and easy setup of the mobile devices
- Accessibility from anywhere in the world
- Lower costs as there's no need to buy new devices
- No device maintenance costs
- Different test types
- Simulated network providers from all over the world
- Good reporting features such as reports, screenshots, and videos

The cons of mobile cloud testing include the following:

- You have less control over the mobile devices.
- Network issues can affect the availability and the functionality of the mobile test cloud.
- Security and privacy issues: Other companies use the same devices, so you need to be sure the app will be deleted in full after the session has been completed.
- Performance problems: Testing the app via the Internet can have an impact on the app execution time and test results.
- Firewall setup: You need to change some of the firewall settings in order to gain access to the development and test environments.
- If the cloud has any system problems or outages, your app test environment may also experience poor performance or outages.

- It is difficult to track down intermittent problems because you have no access to the system.
- Not everything can be tested within a cloud, for example, sensors, interfaces, interrupts, and notifications.
- You have no physical contact with the device.

The following are some mobile cloud testing providers (this list is by no means complete):

- AppThwack (https://appthwack.com/)
- Appurify (http://appurify.com/)
- CloudMonkey (www.cloudmonkeymobile.com/)
- Experitest (http://experitest.com/cloud/)
- Keynote (www.keynote.com/)
- Neotys (www.neotys.com/product/neotys-cloud-platform.html)
- Perfecto Mobile (www.perfectomobile.com/)
- Ranorex (www.ranorex.com/)
- Sauce Labs (https://saucelabs.com/)
- TestChameleon (www.testchameleon.com/)
- Testdroid (http://testdroid.com/)
- Testmunk (www.testmunk.com/)
- TestObject (https://testobject.com/)
- Xamarin Test Cloud (http://xamarin.com/test-cloud)

Important

Mobile cloud testing is a good extension to your in-house testing work but comes with some testing limitations that need to be taken into consideration.

Private Cloud

If the drawbacks outweigh the advantages of a public cloud, but you're still interested in using a cloud testing approach, consider using a private mobile testing cloud. Almost every provider in the preceding list can provide a private mobile test cloud.

A private mobile cloud can be offered as a hosted or locally installed solution. The hosted solution is the more common one because it eliminates the logistics and costs incurred by buying new phones and maintaining devices, such as installing updates and performing configurations. Private cloud providers offer a secure area within the data center that grants exclusive access to the physical devices. On top of that, they offer various security options to meet a company's security policies and requirements.

The locally installed solution is also known as a private mobile test lab. If you decide to use this solution, the mobile cloud provider will provide you with a mobile testing rack including device management software to maintain and allocate test devices within your company to developers or testers. You can also extend the rack with new devices on your own.

This device rack will be located behind your firewall, and devices are connected to the development environment, which alleviates any problems with the speed and connection problems of a public cloud. Security and privacy concerns are also no longer an issue with this solution. By way of example, refer to the mobile test lab from Mobile Labs.[2]

[2]. http://mobilelabsinc.com/products/deviceconnect/

A locally installed solution also comes with the problem that you're responsible for buying and maintaining new devices to be included in the device rack. Furthermore, a private mobile testing cloud can be really expensive as you have exclusive access to the test devices, the cloud vendor provides you with exclusive support, and both you and your company need to train your colleagues to be able to work with the private cloud software and system.

The pros of a private mobile test cloud include the following:

- Easy access to the physical devices
- Fast and easy setup of the mobile devices
- Accessibility from anywhere in the world
- No device maintenance costs (hosted solution only)
- Different test types
- Simulation of various network providers from all over the world (hosted solution only)
- Exclusive access to the test devices
- No security concerns

The cons of a private mobile test cloud include the following:

- It is far more expensive than a public mobile test cloud.
- Network issues can still affect the availability and the functionality of the mobile test cloud.
- Firewall setup: You need to change some firewall settings to gain access to the development and test environments (hosted solution only).
- If the cloud has any system problems or outages, your app will perform poorly and may also experience outages (hosted solution only).
- It is difficult to track down intermittent problems because you have no access to the system (hosted solution only).
- Not everything can be tested within a cloud, for example, sensors and interfaces.
- Additional training is required to work with the cloud provider software and system.

Important

Because of the manual testing limitations, you should consider using mobile test clouds for mobile test automation purposes only with the aim of handling automated testing and fragmentation across several devices. Manual testing should still be performed on real devices in real-world environments and while on the move.

Cloud-Based Test Automation

In Chapter 5, "Mobile Test Automation and Tools," I explained the different concepts of mobile test automation tools. I also described some mobile test automation tools for the iOS and Android platforms. When choosing a mobile test automation tool, check to see if the tool is able to execute the test scripts within a mobile test cloud. Some of the mobile test cloud providers offer an API for various mobile testing tools so you can execute your scripts with their services. This API can help you scale your testing efforts and test different devices in parallel. Some providers offer the possibility of writing test automation scripts on the Web within the cloud testing software.

The advantages of a mobile test cloud also apply to test automation within the cloud. A mobile test cloud can help you create an automated on-demand test environment for your mobile app and mobile platform.

However, using a cloud-based test automation approach also has some additional drawbacks. Test execution on cloud devices is slower than with a local test automation solution, which is partly due to the communication between the cloud provider network and your company network when requesting and sending lots of data. This latency can have an impact on the test results and behavior of your app. Testing the performance with test automation scripts on a mobile test cloud is therefore not an ideal solution. Debugging the test automation scripts on the cloud devices is another issue as script debugging is possible but not yet good enough to work efficiently.

If you're considering using a cloud-based test solution, evaluate several providers to see if they offer the features you need for your mobile app. A cloud-based test automation approach can be a useful addition to your in-house test automation and can make your testing work more efficient.

Summary

In the sixth chapter you have learned about crowd and cloud testing services. Both can be additions to your in-house mobile testing activities but should never be used as the only mobile testing solution for your app.

In the crowd testing section I explained the typical process of a crowd testing cycle. You need to keep in mind that this process will take quite some time for preparation as well as in the bug analysis phase. These efforts should not be underestimated. Furthermore, I described the differences between a private and a public crowd test session. Both approaches differ a lot in the details; however, those details are important in order to avoid infrastructure, data protection, and security concerns.

In the cloud testing section, I explained the features of a cloud testing provider. In addition, I described which test types can be performed within a cloud. In the pros and cons section I outlined possible problems with public and private clouds as well as the advantages.

Chapter 7. Mobile Test and Launch Strategies

So far you've learned a lot about mobile testing and the different test techniques and approaches. This chapter covers mobile test and launch strategies and what you need to include in your strategies. Both the test and launch strategies are very important to every project, and you shouldn't underestimate just how useful it is to keep a written record of them. In the following sections I'll provide you with some examples of mobile test and launch strategies. You'll also find some questions that will help you to write your own strategies.

Mobile Test Strategy

In general, a test strategy is a document that describes the testing approach and work involved in the software development cycle. This strategy can be used to let the project manager, product manager, developers, designers, and anyone else involved in the software development cycle know about the key issues of the software testing process.

A test strategy includes the testing objective, test levels and techniques, resources required to test the system under test, and the testing environment. It also describes the product risks and how to mitigate them for stakeholders and customers. Finally, it also includes a definition of test entry and exit criteria.

The defined test strategy will help, remind, and guide you so you don't forget the important components and features of the app. You really should take the time to write down the steps and resources needed to test the mobile app. Furthermore, the test strategy documents your work and endeavors within the project. Writing a test strategy once doesn't mean that it's then set in stone and you can't make any subsequent changes. On the contrary, it's important that you talk about the strategy with your team from time to time so you can adapt it to any product changes or other more recent circumstances.

However, there is no single mobile test strategy that can be used by every team or for every mobile app as almost every app has different requirements, goals, and target groups and runs on different mobile platforms, which makes it impossible to reuse the entire strategy in every project. But there will, of course, be some items that can and should be reused.

The following sections of this chapter should give you an idea of how to shape a mobile test strategy. You can use them as a starting point and guide for putting together your own mobile test strategy.

Important

Drafting a mobile test strategy doesn't necessarily involve writing endless documentation as you and any other testers simply won't have the time and/or the resources to run through it all, and flexibility is the name of the game in the mobile testing business. A mobile test strategy should serve to guide you and anyone else involved in the project so you can keep tabs on the important parts of the testing process.

Define Requirements

The first thing you and your team should do is define your app's requirements and features at the very beginning of the project. Write them all down and describe the features and possible use scenarios to get a better feeling for the app and its potential users. Rough descriptions of the requirements and features are absolutely fine at this point as they will be specified in more detail during the development process. A written record of these requirements will make it much easier for you to derive your mobile testing strategy.

Here is a list of some possible requirements and features:

- Provide a registration form.
- Provide a login form to access the app's content.
- Provide a logout option.
- Implement a search function within the app.
- The user should be able to create a user profile.
- The user should be able to share content with other users.
- The user should be able to share content on social networks.
- The user should be able to take pictures.
- The app should be available in different languages (English, German, French, and so on).

Furthermore, you should know who's going to use your app. As described in Chapter 3, "Challenges in Mobile Testing," you really need to know your target group and their expectations. Gather as much information as you can to gain important insights into your customers' use scenarios.

Here's a quick recap of possible information about your target group (the full list can be found in Chapter 3):

- Gender
- Monthly income
- Educational background
- Location
- Other apps they use
- Smartphone habits
- Devices they use

Important

If you don't know anything about your target group, check mobile-platform-specific numbers of operating systems and hardware specifications. Furthermore, analyze and gather information about apps that are similar to yours. This is a good starting point to help you gather potential user information.

Based on the requirements and features of your app and knowledge of your target group, you can ask specific questions to collect information for your testing work and scope:

- Is it important to find critical bugs quickly?
- Should the app be tested only in common user scenarios?
- On which mobile platforms should testing be performed?
- What are our customers' carrier networks?
- Are there any areas within the app that are likely to change on a regular basis?
- Is the release (submission) date of the app already known?
- Is there a roadmap for future releases?

Don’t hesitate to ask these kinds of questions as they’re important, and the answers will help you to define your next testing steps and priorities. Don’t worry if you forget to ask a question before you start writing the test strategy —it’s better to ask a question whenever it comes up than not at all.

In the next step you should collect information about the development environment within the company. It is important to know which tools are used to set up a development and test pipeline. Which continuous integration server is being used? Which tools are used to build the app? And which backend technologies are used to process the requests from the mobile app? You also need to know about the architecture of the production environment.

To go about collecting all of this information, you should ascertain the answers to the following questions:

- What kind of software development tools are already available and in use within the company?
- Is there a common build pipeline that must be used?
- Which continuous integration server is used for the project?
- Which tools are used to build the mobile app?
- Which technologies are used to process the mobile requests?
- What skills, such as programming languages, are available within the mobile team?
- How many mobile devices are available within the company for testing?
- What tools and technologies are used in the production environment?
- How many people do we expect to use the mobile app?

The answers to these questions will help you choose, for example, the test automation tool based on the technical knowledge within your team. They will help you to define the test levels and techniques, and furthermore they will provide a first overview of all technologies involved in the mobile project. Knowledge of the development, test, and production environment is very important when it comes to coordinating testing within the project.

Important

Collecting requirements and features is important as you need such useful information for your mobile test strategy. This kind of information is a good starting point for planning your testing activities and will help you to define a testing scope.

Testing Scope

Once you've defined the requirements, you can specify the scope for your test strategy. It's not possible to test an app on every possible hardware and software combination. You should therefore reduce the scope of your testing efforts and initially concentrate on the important parts of the mobile app.

There are four ways to reduce your testing scope:

- Single-device scope
- Multidevice scope
- Maximum device scope
- Use-case scope

Single-Device Scope

The single-device scope focuses on one mobile device during testing. This approach can or will be used if only one device is to be supported by the app or if there is very little time available for the project. In the event of time pressure, you'll probably choose only the most popular device used by your targeted customers. This device will be used for testing with just one mobile carrier network and possibly a Wi-Fi connection. Another approach could be to choose a device from the device group that has older hardware specifications and could therefore cause more problems for the developers in terms of support. You're likely to find more bugs and problems, such as performance or other issues, with this device than with the latest device.

Using only the single-device scope can be dangerous for the mobile app, for the success of the project, or even for the whole company. It is very likely that you'll miss important bugs that occur on other devices and that your customers will submit bad ratings to the app stores. If the app supports only one device—for example, if it is an internal enterprise app—this approach can be used. On the other hand, it's better to test on only one device than to skip the entire testing process.

Multidevice Scope

As the name suggests, the multidevice scope focuses either on several devices from one mobile platform or on multiple mobile platforms. Select the platforms and test devices based on your target group, and then group the devices as described in Chapter 3 in the section "Mobile Device Groups." If you don't have any information about your target group, use the Internet to search for platform-specific numbers and statistics, which will help you select the mobile platform and devices to concentrate on.

A really nice Web page provided by Google is "Our Mobile Planet"[1] where you can get information based on the country, age, gender, user behavior, and behavior during the current year.

1. http://think.withgoogle.com/mobileplanet/en/

Maximum Device Scope

The maximum device scope focuses on as many mobile platforms and devices as possible. This approach can be used for mass-market apps intended to reach as many customers as possible all over the world with no restrictions in terms of platform, device, carrier network, or target group. Testing an app for the mass market is very difficult because there will almost always be a hardware and software combination that doesn't work well with your app, and it's nearly impossible to find this combination. In order to handle and reduce this risk, you need to find a way to test on as many devices as possible.

This approach requires lots of research to gather information and statistics about current device usage, mobile platforms, and operating system versions. It requires information about the different carrier networks and connection speeds from around the world, and so forth.

Once you've collected the required information, consider using a combination of in-house testing, cloud testing, and crowd testing to handle the mass-market situation. Bear in mind that testing just with in-house resources and devices will be either too limited in scope or too expensive.

Use-Case Scope

Besides choosing a hardware testing scope to downsize your testing work, you can also choose a use-case scope to limit the workload. With this approach you can concentrate on certain parts or features of the mobile app and leave out less important ones, such as help texts or edge cases. Nearly every project is under extreme time pressure, thus making it important to define which parts of the app must be included in the testing scope and which can be left out. Write down both parts in your test strategy and describe the use cases that have to be tested and why. If there is enough time, the less important parts should be tested as well.

You may want or need to include the following test scope information in your test strategy:

- Which approach should be used in the project?
- Is it possible to combine approaches for certain parts of the app?
- Why was this approach selected?
- What are the required and main use cases of the app?

Define Test Levels and Test Techniques

Once you've defined the requirements and scope, you need to think about the different test levels and test techniques you want to use in your project. Keep in mind the quality assurance measures overview from Chapter 4, "How to Test Mobile Apps" (the section "Traditional Testing"), when defining them. This overview will help you to derive the test levels and techniques for your mobile app.

Test Levels

As shown in Chapter 5, "Mobile Test Automation and Tools," the focus of test levels from non-mobile software development tends to shift throughout the mobile software development process. The "mobile test pyramid" shows that the unit testing level is the smallest part compared to end-to-end testing, beta testing, and manual testing.

In most software development projects, the developers are responsible for writing unit tests. This is also the case with mobile app projects. Mobile testers are responsible for writing the end-to-end test automation including integration testing. However, everyone on the team should be responsible for the app’s quality—they should all support the mobile tester in his or her work.

As mentioned in several chapters in this book, manual testing is a very important part of a mobile development project and forms an essential part of the test levels in a mobile project. However, there are other levels that are important for manual testing: acceptance, alpha, and beta testing. User acceptance tests can be performed to test the mobile app against the user requirements to verify that the app covers them all. This step is usually carried out by a tester, product manager, or the customer.

In mobile development projects, alpha and beta testing are important test levels that should form part of your test strategy. Whenever a feature is implemented within your app, you should test it with potential customers to gather early feedback about it. If testing with potential customers is not possible within alpha tests, you could try testing the feature with your work colleagues to get feedback from outside of the development team.

Once the mobile app has reached a defined maturity—for example, all of the features have been implemented or only two bugs were found in the last week—you should consider using beta distribution tools or a crowd-based testing approach to perform beta tests with potential customers. The criteria that determine that a testing phase is finished and that the next one is ready for execution also need to be documented in the test strategy.

Typical software test levels that should be used in a mobile development proj-ect are listed here:

- Automated testing
  - Unit testing
  - End-to-end testing (including integration testing)
- Manual testing
  - Acceptance testing
  - Alpha testing
  - Beta testing
- Regression testing

Once you’ve defined the test levels, you should also consider defining how intensively each level should be tested from a functional and nonfunctional point of view. Please keep in mind that not all test types will be included at every test level.

Furthermore, it can be helpful to define some metrics that measure the current state of the application. Possible metrics may include the following:

- Each feature must have at least one unit test.
- Each feature must have at least one end-to-end test.
- There should be no warnings or errors in the static analysis check.

Functional testing should include the following:

- Identify the functionality of the mobile app.
- Test the different parts against the functional requirements.
- Define and execute the test cases.
- Create input data based on the specifications.
- Compare the actual and the expected outputs.

Nonfunctional testing should include the following points:

- Load testing
- Performance testing
- Usability testing
- Security testing
- Portability testing
- Accessibility testing
- Internationalization/localization testing

The following test-level information could be used in your test strategy:

- Which test levels will be used in the project and why?
- Which parts are to be used for functional and nonfunctional testing?
- Define and describe the automated test levels. Which parts need to be unit tested and which parts will be tested using an end-to-end test automation tool?
- Define and describe when the software has reached a certain maturity and can be used for alpha and beta testing.
- Define and describe metrics relevant to the project.

Test Techniques

You can draw upon quality assurance measures (Chapter 4) to define your test techniques and methods. Consider using static and dynamic approaches to test your mobile app from different points of view.

As described in Chapter 4, I recommend using static code analysis tools in your static testing approach to test your mobile app's code for any bugs or problems. Keep in mind that the app code is not executed during static testing. Furthermore, all of the project documentation should be reviewed for completeness.

With the dynamic testing approach you should use white and black box testing techniques to test your app. White box testing should be done by the developers and covers the following:

- Statement coverage
- Path coverage
- Branch coverage
- Decision coverage
- Control flow testing
- Data flow testing

Black box testing should be done by the software testers including:

- Equivalence classes
- Boundary value analysis
- Decision tables
- State transitions
- Cause-effect graph

However, the developers should also test for boundary values and state transitions at the unit testing level in order to be sure that each unit correctly handles those situations.

Your test strategy should also include a written record of which test technique is to be handled by whom.

Besides the aforementioned techniques, you should consider using exploratory and risk-based testing to organize and downsize the testing work within your mobile team.

Important

Define test levels for your mobile app based on its features and requirements. Quality assurance measures will help you define your test methods and techniques.

The following information about test techniques could form part of your test strategy:

- Which test technique will be used with your project and why?
- Define and describe the order of the test techniques; for example, static code analysis and document review are to be followed by white box testing, black box testing, and then exploratory test sessions involving the whole team.
- Which team members are to apply the different test techniques?
- Define and describe the manual testing process, such as acceptance testing, exploratory testing, alpha and beta testing.
- Define and describe the test exit criteria for white and black box testing, for example, 80% branch coverage with white box testing.

Test Data

Nearly every app processes, creates, and sends data from the mobile app via data networks to different backend systems. The processed and transferred data differs from mobile app to mobile app and has differing requirements and complexities. One component of your mobile test strategy should be the required test data. It is important to define the test data as realistically as possible based on the features and requirements of your app.

The following three points are an example of different test data types:

- **Configuration data:** This data includes configurations for the mobile app or for the backend system. It could, for example, include decision rules for rules engines and/or settings for databases and firewalls.
- **Stable data:** This usually involves data with a low rate of change and a long duration. A typical example of this is customer information such as username and password, or product information.
- **Temporary data:** This kind of data is likely to change frequently, meaning that it can be used only once or will be created while the app

is running, for example, payment details or vouchers.

Once the test data requirements are clear, find a way to save the test data so you can re-create and reset the data in a defined state whenever you need it. One possible solution for this is a database where the stored information can be used during the development and testing process. Another advantage of this is that you can also use the database for manual and automated testing. On the other hand, you can use test data management tools to organize data within your project.

Once you've set up the reset and re-creation process, you should start creating the data as soon as possible as this will help you and your colleagues during the app development process.

Depending on the app, you may need lots of test data. If this is the case, it might be a good idea to use a generator to create the data automatically. If a data generator is used, it is important that the functionality and necessary parameters be documented.

If test data is available within your project, don't forget to adapt it to new features and changes. It is very likely that over time features will be improved and the test data requirements will change. Your test strategy should therefore also outline a process for updating the test data, including responsibilities and triggers for the update process. It is also recommended that you define a strategy for how outdated test data should be archived in order to reproduce incoming bugs in old features or versions of your app.

The following test data information could form part of your test strategy:

- How is the test data generated?
- Where is the test data stored?
- How is the documentation handled, for example, test data together with test results?
- How often will the test data be updated?
- Who is responsible for the test data?

Select Target Devices and Test Environment

Now that you've described the features, requirements, test level, and technique as well as the test data, you need to think about the test environment and test target devices for your strategy. As you learned in Chapter 3, grouping your test devices or mobile Web browsers is a good approach to determine which devices should be used for the mobile app. Creating such mobile device groups requires information about your target customer group and their usage scenarios. Don't forget the testing scopes described in this chapter when doing so.

Once the device groups are in place, you'll need to select devices from those groups to have them available within your team. It is recommended that you have at least one device from each group available for testing. However, I recommend at least five devices from each group in order to have a broader mix of hardware and software combinations with different form factors and displays. You should also keep a record of why you chose those test devices.

Now that you know which devices are needed for testing, you need to buy or rent them. Buying all of the devices you need can be expensive and perhaps is not an option due to the project's budget. A good approach to save some money is online auctions where you can buy used devices. In most cases the devices are in good enough shape for your testing work.

If buying is not an option at all, you can rent the devices. As mentioned in Chapter 3, there are several mobile device lab providers on the market that will lend you the devices you need for a set period. However, check the rental prices and compare them with the device price. If you want to rent the devices for a prolonged period, renting will probably be more expensive than buying the device in the first place.

Another alternative is Open Device Labs[2] where the devices can be borrowed for free. Check the Open Device Labs map to find one in your area. If you want a free device, you can also ask around at your company or even see if someone in your family has the device you need that you can borrow for a while.

2. http://opendevicelab.com/

Once you've specified your device strategy, you need to think about the test environment: the backend systems. You need to know the architecture of the backend systems such as databases, payment systems, APIs, and any other kind of system involved in the mobile project. If you have systems information, you need to be sure that it’s also available within the test environment so you can test as though you’re in the production environment. If your test strategy contains in-the-wild or on-the-move tests, there’s an additional requirement: the test environment must be accessible from outside the company network.

The following information about the target devices and the test environment could form part of your test strategy:

- Which devices will be used for testing?
- Why are those devices used for testing?
- Are the devices available within the company or do you need to acquire them?
- What are your reasons for choosing those test devices?
- What are the requirements for the test devices and the test environment?
- Is there an update policy for the mobile devices?
- When will new devices be integrated into the development and testing process?
- What are the usage scenarios of the system?
- What does the backend system consist of?
- Is the test environment similar to the live environment?
- Can the test environment be used for testing purposes from outside the company network?

Manual and in-the-Wild Testing

As you have learned so far, mobile testing requires lots of manual testing and testing in real-life environments. Think of the example with the ski and snowboard app from Chapter 1, “What’s Special about Mobile Testing?,” where testing on a mountain is required to see if the mobile app actually works under real conditions.

Manual testing in the wild is essential for your mobile app and requires lots of planning beforehand to avoid useless test scenarios while testing on the move. You should therefore try to identify common real-world usage scenarios for your mobile app and its features. Write the scenarios down in your test strategy and rank them based on priority and importance to your project.

Furthermore, it is recommended that you define test routes, such as by bus, train, car, plane, or while walking. Within these routes, describe possible scenarios that should be tested. These routes allow you to simulate real mobile users while they commute to work or while they're traveling around.

Last but not least, you should define data network scenarios based on your target group. After gathering information about your target group, you will know in which regions they live and what kind of data networks and speeds are available, for example, 4G, 3G, or EDGE. Based on that information, you can limit app testing to those network speeds, but don't forget to test on different network providers.

Here are a few example usage scenarios:

- Test your app based on its features, such as outside in sunny places or inside an office.
- Test to see if the app can be used in different weather conditions.
- Use multiple apps such as e-mail, chat, and news while your app is running in the background. Check to see if the app is influenced by other apps.

Here are some examples of route scenarios:

- Use the app while commuting to work by train, bus, or car.
- Use the app while running and check the behavior of the sensors.
- Use the app while walking through a city or the countryside and check the sensors and the GPS or compass.

Here are some examples of data network scenarios:

- Test how the app works in fast data networks like 4G or 3G.
- Test how the app handles the data network change from 4G to 3G or even 2G.
- Test how the app handles packet loss or a complete loss of network.

In-the-wild testing requires lots of movement and is a challenging task that needs to be handled during the app's development process. If you don't have the time or option to test your app in real-world scenarios, think about using crowd testing in tandem with your in-house testing work. However, remember the pros and cons of crowd testing from Chapter 6, "Additional Mobile Testing Methods," as they can have an impact on your project planning, coordination, timing, and budget.

Don’t write complex test cases and scenarios with exact steps for in-the-wild testing. While you’re on the move, you’re unlikely to have a laptop with you to check the test cases. This is inefficient because it will destroy the real-life testing scenarios, behavior, and user simulation. You’ll probably have a bag with you containing lots of devices for your in-the-wild testing. When planning such scenarios, keep them short and informative so that they can be used while on the move. I recommend that you print out the scenarios or write them down on a piece of paper so you can just read them and keep them in mind.

The following information about in-the-wild testing could form part of your test strategy:

- Define and describe the usage scenarios of your customers.
- Define and describe the test scenarios in the wild.
- Define and describe the different data networks that need to be used for testing.

Mobile Checklists and Tours

You can add mobile checklists and mobile tours to your mobile test strategy. As described in Chapter 4, mobile checklists can be very important for your mobile app as they help you keep a record of things that can’t be automated or are likely to change on a frequent basis. If you know the requirements and features of your app, you’ll probably also know the parts of the app that need to be tested repeatedly. In that case it’s good to add those features to a checklist as part of your test strategy.

Based on the features of your app, it’s useful to define mobile testing tours to concentrate your testing efforts on special parts of your app. Cem Kaner[3] describes a tour as “. . . an exploration of a product that is organized around a theme.”

3. http://kaner.com/?p=96

Using tours in your mobile testing work helps you to explore and understand how the mobile app works. It also helps you come up with new test ideas while you’re testing the app. Please refer back to Chapter 4 for a description of some tours backed up with mnemonics.

Here are some examples of testing tours:

- **Feature tour:** Explore and test all of the possible features within the app.
- **Configuration tour:** Explore and test every part of the app that can be configured.
- **Gesture tour:** Use every possible gesture on every screen to see how the app handles the different inputs.
- **Orientation tour:** Change the orientation of every screen from portrait to landscape and vice versa to see if any problems arise.

I use the following mobile mnemonics in my projects:

- FCC CUTS VIDS (http://michaeldkelly.com/blog/2005/9/20/touring-heuristic.html) from Michael Kelly
- I SLICED UP FUN (www.kohl.ca/articles/ISLICEDUPFUN.pdf) from Jonathan Kohl

The following information about mobile checklists and tours could form part of your test strategy:

- Which checklists will be used in the project and why?
- Describe the used checklists and tours.
- Describe when the checklists and tours should be used and by whom.

Test Automation

Test automation can also form part of a mobile test strategy. Depending on the mobile app and its lifecycle, you may not need to automate it. If that's the case, it's important that you document and describe the reasons why test automation is not necessary.

However, if your app requires test automation, you should start to think about automation and the tools you want to use as soon as possible. Think back to the different mobile test automation concepts in Chapter 5 and their pros and cons. Choose the tool that best fits your current project situation and where you have already gained some experience with one of the tools or programming skills, or where you have a mobile testing setup or environment in place. This will save you lots of time and money.

Describe the mobile test automation tool in your test strategy and why you chose it for this project. Based on the described features and requirements, you can define the parts of the app that should be automated and which do not need to be.

During the next step you should describe the devices on which the automated test has to run and in what environment the tests should be executed.

Your mobile app project should find the right balance between real devices and virtual devices if you're not able to test everything on real devices. A mixture of real and virtual devices can also be more cost-effective.

Once you've defined the devices and test automation environment, I recommend that you also define test suites to group the different automated test cases based on common features, areas, and requirements within your app. With the help of the test suites you can decide which tests should be executed when and how often. For example, you can define a smoke test suite containing several automated tests from each part of the app to make sure a commit to the central code repository didn't break anything. This test suite should be small and run every time the code changes in order to receive fast feedback and to swiftly inform the developers of any problems.

Here are some examples of test suites:

- Smoke test suite containing a few test scripts to check the basic functions of the app and to provide fast feedback
- Medium test suite or a test suite containing only specific functionality
- Full regression test suite containing all test scripts to be executed once a day or during the night
- Test suites containing only user scenarios such as the registration or checkout process

Test suites can be a great way to create a balance between fast feedback and broad test coverage.

When the test suites are in place, define when they should be executed; for example, the smoke test suite must be run after every commit by the developer. The medium test suite could be executed every two hours. The full regression test suite should be executed once every night.

You should also define where the test automation should be executed. One such solution can be where the unit tests are executed on the developer's local environment before changes are pushed to the central repository. The unit tests and end-to-end tests can be executed on the CI server.

The following information about test automation could form part of your test strategy:

- Is a CI server used for the build process?
- Which CI server is used and why?
- Which test environment is used to execute the test automation?
- Which test automation tools are used in the project?
- Are the tests executed on real devices and virtual devices?
- Which devices must be connected to the CI server and why?
- Is a cloud provider used for the test automation?
- Define and describe the test suites and groups.
- Define and describe the different build triggers and execution times.
- Define and describe where the different test automation suites will be executed, for example, everything on a local developer machine or only on the CI server.

Product Risks

Every project is exposed to different kinds of risk. It is important to work out both project and feature risks so you can then define possible solutions for them. Consider both the likelihood of risk occurrence and risk impact. If the product risks are clear, you can start implementing a risk analysis approach that can be used by the whole team when defining, implementing, and testing new features.

The following information about the product risks could form part of your test strategy:

- Which parts are critical to the business?
- What is the likelihood of the product risks occurring?
- How should the business-critical parts be tested?
- What is the potential impact if a critical problem occurs?
- How is the feature risk analysis performed?
- Is there a disaster plan in place?

Important

Creating a mobile test strategy is not easy as it needs to cover lots of mobile testing information. Your strategy may also need to be modified during the development process because of changed product features or priorities.

Mobile Launch Strategy

As important as the mobile test strategy is, it's also really important to write down a mobile launch strategy. Launching an app is not easy, and lots of problems can occur during and after it has been launched. This part of the chapter will describe the important pre- and post-launch activities for a mobile app.

Pre-Launch—Check the Release Material

As mentioned in Chapter 4, you should think about putting together a release checklist to make sure that all the release information is available and in place. You should also perform the update and installation test before submitting the app to the app store.

The tests you perform before committing the app to the app store shouldn't be limited to the app itself. Your launch strategy should outline if and how the backend services are tested before a new release.

Ask yourself the question "Does the new version of the app require some new backend services or API calls?" If so, are those services or API calls already live in the backend system? If not, it's very likely that your app will be rejected by the app store vendor. If backend services are available, check the new and existing features of the app in the production environment one final time.

When the app is ready for launch, check the release notes and feature description in the new app store information material. Read through the texts and compare them with the new and existing features. In the release notes it's important to be specific about the new features; describe them well and explain them to the users. It is also important to provide the release notes and app description in every supported language.

Don't forget to check the screenshots of the app. They should be in the same language as the notes, be the same size as the previous screenshots, and have the same status bar icons showing the time, battery, and network state. Figure 7.1 and Figure 7.2 provide examples of what not to do: the status bar contains different icons for the same app.

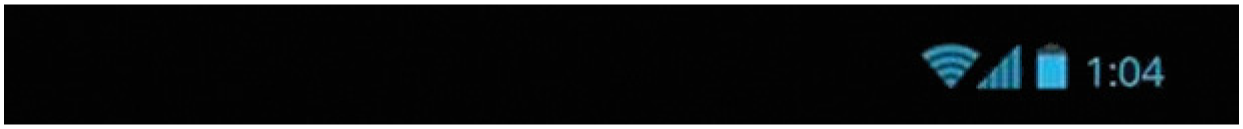


Figure 7.1 *App store screenshot*

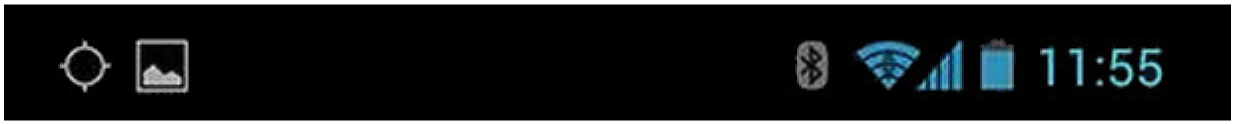


Figure 7.2 *App store screenshot of the same app with different status bar information and sizes*

The app store screenshots need to have the same status bar and the same look-and-feel in order to portray a professional image.

If the new features are described by means of a video, watch the video again and listen to the feature description and information it provides. Ask if there is more marketing material available that needs to be checked.

Last but not least, there is one really important point: don't release your app to your customers on a Friday evening or just before you leave the office. In most of the app stores it takes some time (up to several hours) before your app is listed and available for download. If the office is empty, no one can react to any critical bugs or release problems that may occur right after the release.

Some app stores don't have an app submission process and can therefore react quickly to problems by releasing a hot-fix version of your app right after the last release. When possible, release your app either in the morning or on the weekend so that you have some time to react to critical issues.

The following information about the release material could form part of your launch strategy:

- Define and use a release checklist.
- Perform the update and installation test again.

- Check to make sure all of the backend services and APIs are available on the production environment.
- Define which new and old features should be checked again on the live environment.
- Define and describe how the app store material, including release notes, screenshots, and videos, should be checked.
- Define and describe when the app should be submitted to the app store.

Post-Release—What Happens after an App Launch?

The post-release phase starts once you've released your mobile app. During this phase you and your team should pay attention to several points to get feedback from your customers and to handle any questions or problems they send you.

The first thing you should do right after releasing your app is to download it and install it from the app store to make sure it's working as expected. If the version is OK, you should archive it on a file server to be able to install it again later, for example, to reproduce problems or bugs that may be reported. This step can also be handled by your build pipeline within the continuous integration system.

However, there are a few more issues that you, your team, and your company need to handle. The following sections describe the activities that can be performed after releasing your app in order to get further information about your customers and any potential problems. It is important to define and write down such items in your app launch strategy.

Community Support

Providing community or customer support for a released product is essential, especially if you're offering a paid app. Whenever users have a problem, they should be able to contact someone at your company or the mobile team to get answers to their questions or problems. If no one looks after the users or communities, they will write bad reviews or simply stop using your app.

You should therefore make sure you man every possible social media channel and check for customer feedback, questions, or problems that need to be looked into. It is important to respond to such queries to give customers the feeling that someone is listening to them. It is also possible to select some users and ask them questions about the new version of the app. The gathered feedback can then be used to improve the app in future releases.

> **Important**
>
> If your company has a customer support department, I recommend that you spend a bit of time with them to get a feeling for customer needs and problems.

Reviews

The next channel you have to monitor after releasing your app is the app store reviews. Read them carefully to get some feedback and bug reports from your users. However, I highly recommend that you handle those reviews with care. There are lots of people out there who like to write negative feedback about apps that is not true. Whenever a user complains about your app, try to reproduce the problem in order to be able to create a bug report aimed at fixing the issue in a subsequent release. If you’re not able to reproduce the issue, try to reply to the reviewer to ask specific questions in order to get more information.

However, not all app stores provide a reply function within their review section so that you can talk to your customers. If the app store doesn’t provide such a feature, you can write your own review, stating that you are the developer, tester, or product manager of the app and would like further information or are offering some sort of solution to the problem. However, be sure you’re allowed to write reviews within the app store to reply to previous reviews. Please refer to the app store review guidelines to make sure you’re not violating any rules. If you’re able to reply to reviews, use this functionality to interact with your customers and to learn from them.

If you get lots of negative feedback due to a misunderstanding within the app or its features, you can also provide some sort of troubleshooting guide or tutorial within the app description. However, if you get lots of negative feedback about your app because the customers didn’t understand the feature, you need to rethink the whole app or feature, perform additional usability testing, and provide an update as soon as possible.

Crash Reports

Another valuable source of information is the crash reports of your app. If you implemented tools like HockeyApp,[4] crashlytics,[5] or TestFlight,[6] you should check them after the release to see if there are any problems with your app. Not every problem that occurs during runtime will end up being an app crash. If your app has good exception handling in place, errors will not be shown to the user but will be captured by the crash reporting tools. This kind of information is very important to help you improve the app in further releases.

4. http://hockeyapp.net/features/
5. http://try.crashlytics.com/
6. www.testflightapp.com

The tools provide a Web interface where the top app crashes are ranked, grouped, and categorized. They show the total number of crashes and the number of users affected by these problems. Nearly every crash reporting tool provides nice graphs showing, for example, app distribution and app crashes over time. Some of the crash reporting tools provide an option to send feedback from within the app to the crash reporting backend. Furthermore, some of the tools offer third-party integration for a bug tracking system.

If you haven't implemented a crash reporting tool, some app store manufacturers provide basic crash reporting functionality that can be used as a starting point.

Important

Implement a crash reporting tool as it will help you and your team to get more insight into the problems and crashes within your app.

Tracking and Statistics

To gather information about your customers and their usage of your app, you should implement some sort of tracking mechanism to collect important data. This kind of information will be aggregated by tracking tools to generate statistics about your app and feature usage. If your app uses a tracking mechanism, check the statistics generated post-release.

Depending on the tracking implementation, you can get information such as:

- Mobile operating system version
- Mobile device manufacturer
- Device model
- Display size
- Mobile browser version
- Number of page views
- How often a certain feature was used
- How often the registration process was aborted

With the help of those statistics and numbers, try to understand your users' behavior so you can tweak the app and its features. The following list is a sampling of some mobile tracking tools:

- adjust (www.adjust.com/)
- appsfire (http://appsfire.com/)
- AppsFlyer (www.appsflyer.com/)
- Clicktale (www.clicktale.com/)
- iMobiTrax (www.imobitrax.com/)
- MobileAppTracking (www.mobileapptracking.com/)

As you can see, it's not easy to create mobile test and launch strategies as both require and contain lots of information about testing topics and pre- and post-launch activities. Please keep in mind that such strategies are not set in stone. You should rework and adapt both strategies whenever changes occur in terms of the product, risks, or any other priorities. Mobile test and launch strategy documents are an ongoing process, and every team member should be responsible for updating and extending them.

Summary

The main topic of Chapter 7 was the mobile test and launch strategies. Every mobile team needs test and launch strategies in order to remember important tasks before and after the release of the app.

When establishing a mobile test strategy, the following topics should be covered and defined:

- Requirements definition

- Testing scope
- Test levels
- Test techniques
- Test data
- Target devices and environment
- Test automation

With the questions provided in this part of the chapter you are now able to set up your own mobile test strategy for your app or for your company.

The other part of this chapter covered the mobile launch strategy. The importance of the release material, including feature descriptions, screenshots, and any other marketing material, was described. With the help of the questions provided, you should find it very easy to check whether everything is available for the release.

After the release of the app it is important to have community support in place, where users can ask questions when they have any kind of problem with your app. Furthermore, the importance of crash reports, tracking, and user statistics was established.

Chapter 8. Important Skills for Mobile Testers

Software testers and mobile testers in particular are facing more and more requirements as mobile apps are becoming increasingly complex and the time to market is getting shorter. Mobile testers need to be able to efficiently test complex mobile applications within a very short period of time to deliver excellent products to customers. Besides testing knowledge, mobile testers must have several other important skills to manage the complexity of the systems and the huge number of different scenarios.

This chapter is all about software testing skills and how to improve your mobile testing skills to become a better mobile tester.

Skill Set of a Mobile Tester

Besides knowledge and skills regarding software testing methods, approaches, mobile apps, and devices, testers must have a solid basic set of soft skills if they want to be successful in the mobile development business. In the following sections of this chapter you'll read about various skills every software tester should have.

Communication

The ability to communicate is one of the most important skills a software tester must have. Software testers must be able to describe their testing work to different kinds of people at different levels within the company. They need to be able to talk to developers, designers, product managers, project managers, other software testers, and management as well as to customers. Talking to other software testers and developers requires technical skills and detailed knowledge of the different parts of the software. Reporting bugs and showing others their mistakes can quickly lead to negative emotions, which is why it's important to report bugs in a clear and concise way without bringing emotions into the conversation.

Talking to product managers, project managers, designers, or management requires the ability to describe problems and bugs to nontechnicians at a higher level in a way that is clear and understandable.

In addition to verbal communication, software testers need strong written communication skills as they have to be able to describe problems and bugs in such a way that every potential stakeholder can understand them.

Listening is also an essential part of communication, and software testers must be able to listen carefully when other people are talking and describing their thoughts or problems. It is important that you don’t interrupt other people when they’re talking. If you have any questions, make a note of them and ask them once the person has finished.

To help you improve your communication skills, I recommend that you read, read, and read—not just books but blogs, newspapers, and any other kind of material that will help you improve and expand your vocabulary, especially if you’re not communicating in your native language.

I also recommend that you watch movies or TV series in other languages to bolster your vocabulary. If you have more time available and want to spend some money, you can attend language and communication classes, which are a great way to improve your language and communication skills.

Last but not least, I recommend that you give talks at user groups, conferences, or within your company. Such experience will have a major impact on your communication skills because you get feedback from your audience right away.

Poor communication generally leads to disagreement and misunderstandings, which can be avoided by following some simple rules:

- Listen carefully.
- Don’t interrupt other people while they’re speaking.
- Don’t speak too loudly.
- Don’t speak too quickly.
- Speak clearly and precisely.
- Make eye contact with your audience.
- Don’t get personal when communicating with other people.
- Be able to communicate on different levels, ranging from technical to nontechnical audiences.
- Improve your vocabulary by reading books, blogs, and newspapers.

Important

Software testers need to be diplomats, technicians, and politicians all rolled into one as they have to be able to talk and listen to different stakeholders within the company.

Curiosity

It's human nature to be curious, and software testers need to be curious to explore, discover, and learn new things about the software they're testing and the product domain. A curious software tester explores the software to get as much information as possible out of the system to identify potential problems and raise interesting questions about the product. It's important to go beyond the usual software testing approaches and methods to discover new things.

To be able to discover new things, it's important to be open to new technologies and willing to try new approaches and methods. A curious software tester doesn't rely on statements from other people; he or she questions them to gain more information.

If you'd like to train your curiosity, I recommend that you download a random mobile app or software application and start exploring its features. Try new approaches and methods while you're exploring the software. Try to break the system and start questioning the features. Make note of everything that feels wrong or evokes some sort of strange reaction during your exploration so you can raise questions or point out any possible problems with the software.

Important

Be curious; explore and discover every part of the software to raise problems or questions. Don't rely on statements from other people; question them.

Critical Thinking

Another really important skill every software tester must have is critical thinking. With the help of critical thinking good software testers are able to see the larger context of the software and its features. They are also able to break down the software or the requirements through analysis and reflection. This is very important to gain a deep understanding of the product and to focus on the right testing work.

The following quote from Michael Bolton describes critical thinking in a nice way: “Critical thinking is thinking about thinking with the aim of not getting fooled.”[1]

1. www.developsense.com/presentations/2010-04-QuestioningBestPracticeMyths.pdf

It’s important to question your own thinking, testing methods, and approaches as well as your own decisions and the software that needs to be tested. Ask yourself the following questions:

- What is the problem of this feature/software?
- Is it really a problem?
- Why have you tested this feature that way?
- Have you thought about this?
- Are you sure about this?

A very good three-word critical thinking heuristic from James Bach[2] is Huh? Really? So? Each word suggests an investigation pattern that indicates assumptions, sloppy inferences, and misunderstandings:

2. www.satisfice.com/

- **Huh?**
  - Do you understand what others are talking about?
  - Is it confusing?
  - Is it vague?
- **Really?**
  - Is it factually true?
  - What evidence do we have for it?
- **So?**
  - Why does this matter?

- To whom does it matter?
- How much does it matter?

Use this critical thinking heuristic in your project and start questioning your own work and the mobile app. For further information about critical thinking, take a look at the slides from Michael Bolton's course, Critical Thinking for Testers.[3]

3. www.developsense.com/presentations/2012-11-EuroSTAR-CriticalThinkingForTesters.pdf

Tenacity

Reporting bugs or raising issues can be exhausting and difficult. Not every issue found by a software tester will be fixed. The issue may not be important enough for other team members, or perhaps there is not enough time to fix it. It is part of the software tester's job to be tenacious and fight to get bugs resolved. If the software tester thinks a bug may be critical for the customers or the system, he or she needs to initiate a discussion to describe and explain why it needs to be fixed in the next release. The keywords here are "Bug Advocacy." The Association for Software Testing provides a training course on this important topic.[4] If you want to get a first impression of Bug Advocacy, take a look at the slides from Cem Kaner.[5]

4. www.associationforsoftwaretesting.org/training/courses/bug-advocacy/
5. www.kaner.com/pdfs/BugAdvocacy.pdf

High stress levels are common before a release and often cause developers or project managers to neglect the agreed-upon software quality standards. In such situations software testers must be tenacious and explain or raise bugs over and over again until the software quality standards have been met. But be careful with this as you may end up being considered a nuisance. Here it's important to rely on your strong communication skills.

Software testers have to be tenacious while testing software such as mobile apps. Depending on the kind of app being tested, such as a game, it is very likely that certain game levels have to be tested over and over again in order to be sure that each level works as expected. This can also be very exhausting but requires tenacity or some test automation.

Important

Be tenacious during testing and during possible discussions about bugs and errors within the application.

Constant Learner

The current mobile world and technology are changing rapidly. To maintain pace with this environment, software testers and, in particular, mobile testers must be able to adapt and learn new things really quickly. Software testers need to take note of changes taking place around them in order to adapt and learn new approaches, methods, and technologies.

To keep pace and learn new techniques and tools, software testers can read blogs or books and attend conferences and training courses. On the other hand, it's important that software testers be able to learn during their daily job, while testing software, and while using tools such as test automation tools. Whenever a new tool, technique, or technology enters the market, every software tester should be motivated to gather information about these new items and to learn about them.

Important

Learning and improving personal skills should be a lifelong habit.

Creativity

Another important skill a software tester should have is creativity. It is important to be able to generate creative ideas to test software in very different ways so as to find more bugs and provide the team with useful information. The creativity process starts with designing the test cases and test data. Software testers need to think in different ways to find every conceivable use for a piece of software.

When the default testing approach is complete and there's some project time left for testing activities, I recommend that you test the software again from a completely different point of view; for example, walk through the bugs again to generate new testing ideas, or talk to colleagues or beta testers to get new ideas for your testing work. Try to be creative with your data inputs, when using the navigation, or anything else that comes to mind. You'll be surprised about the results from that kind of testing and will no doubt come across some more bugs.

Important

Mobile testers in particular have to be creative in order to use mobile devices in different ways by paying attention to all the interfaces, sensors, and locations.

Customer Focus

Every software tester should have a strong customer focus. It's important that software testers try to think like a customer in order to determine whether the software being tested meets customer needs. Testers therefore need to have lots of passion and determination and be able to identify strongly with customers.

A strong customer focus requires you to be a product and field expert within your team. You also need to have an overview of released features and functionality and be able to keep an eye out for future releases. It is very important to be aware of customer behavior in order to know which features and functionality they use. If possible, software testers should talk to customers to determine their needs and problems. This can be a challenging job, so software testers have to be patient.

When software testers have a strong customer focus, they can contribute their knowledge to every phase of the software development process, which in turn helps to build better products. To help improve customer focus, I recommend spending a couple of weeks working with your customer support department to get a better feeling for customer needs.

Programming and Technical Skills

The fact that software products and mobile apps are becoming increasingly complex leads to the challenge that mobile testers also need to have solid programming skills as they help software testers to understand the system under test, to communicate with developers at code level, to review code from developers or other software testers, and to write test automation code which is now becoming essential in every project.

Mobile testers with no programming skills need to train themselves by reading a book about programming languages or patterns, by following a programming tutorial on the Internet, or by attending a programming course. It's also possible to ask a developer if he or she can train the mobile tester within a project or company.

Thanks to programming skills, mobile testers are able to write test automation code from unit to end-to-end level. They are able to attend code reviews to ask technical questions and are likely able to write shell scripts in order to automate either a build pipeline or any other task that the team needs to perform.

Besides coding skills, every mobile tester must be able to understand technical system architectures in order to be able to ask critical questions about the architecture and to know how to test every part of it.

Important

Every mobile tester needs programming skills in order to be able to write test automation code and to be able to attend code reviews and technical discussions.

How to Improve Your Mobile Testing Skills

As mentioned in several parts of this book, the mobile world is changing rapidly, so you'll need to hone your skills every day to keep pace with the mobile testing world. You have to constantly learn new things to generate new testing ideas, to collaborate with developers during their work with programming skills, and you'll also need to understand customer needs.

In order to improve your mobile testing skills it is important to have at least one mobile device available. In most cases this is your private and personal device. If possible, I recommend that you have several devices available at home with different mobile platforms so you can learn everything about those platforms. You don't need to keep buying the latest devices; you can settle for used phones or even older versions to learn about each platform. If you're not in a position to buy lots of devices, keep in mind the Open Device Labs, where you're able to borrow different devices for free.

Learn from Other Apps

A very easy way to improve your mobile testing skills is to learn from other apps. I recommend installing and using as many apps as possible from different categories within different app stores to see how they work and behave. Check how other apps have implemented their navigation and update mechanism and how they use mobile-specific features such as the camera or other sensors.

However, besides using them, it's important to check the update texts of those apps. I recommend that you uncheck the automatic update functionality of all of your apps so you can install new versions manually. Before pressing the update button in the different mobile app stores, read the update texts and app descriptions carefully. There are lots of companies or developers who are really precise when describing what the new version of the app is all about. They describe which bugs are fixed with the new version and which new features have been added.

If there is a bug description in an app's update text, try to reproduce the bug so you can see it with your own eyes. This can be a lot of fun, but you may find it takes a while to provoke the bug. But this is in itself a great way to learn lots of new things.

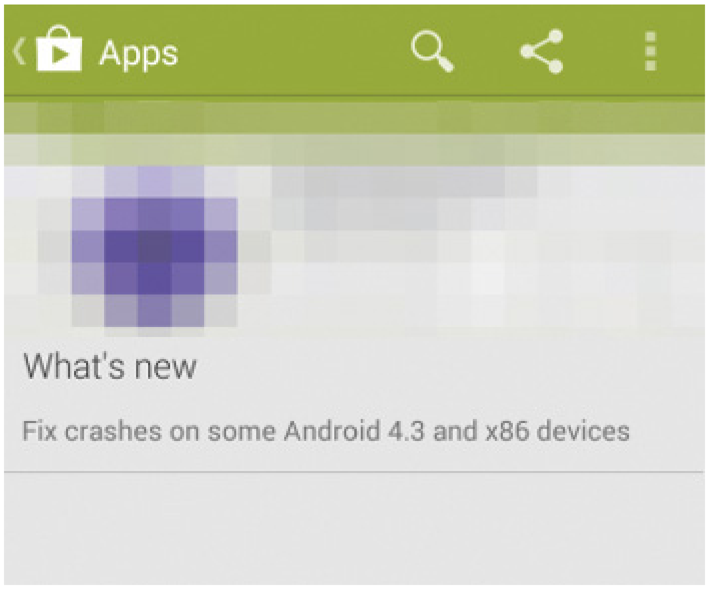
You will probably get new testing ideas, come across new ways to use an app and new approaches to provoking a bug, and learn things that you may never have thought about before. The following sections will provide some examples involving different kinds of bugs and descriptions from mobile apps I check from time to time.

Important

The app screenshots in the "Crashes on Specific Devices" section are anonymized. All of the examples are based on the Google Play store. However, the same sort of bug and feature description can be found in every other mobile app store.

Crashes on Specific Devices

The app in Figure 8.1 crashed when using Android version 4.3 and on some x86 devices. If you want to reproduce this crash, you'll need to get a device with Android 4.3. If such a device is available, find the section or app view that's crashing. This may prove to be rather difficult as there are lots of devices with Android 4.3 available and every device can behave differently. If you manage to find the crash, try to understand why it happened. Maybe it's due to a poor Internet connection or just a bad implementation.

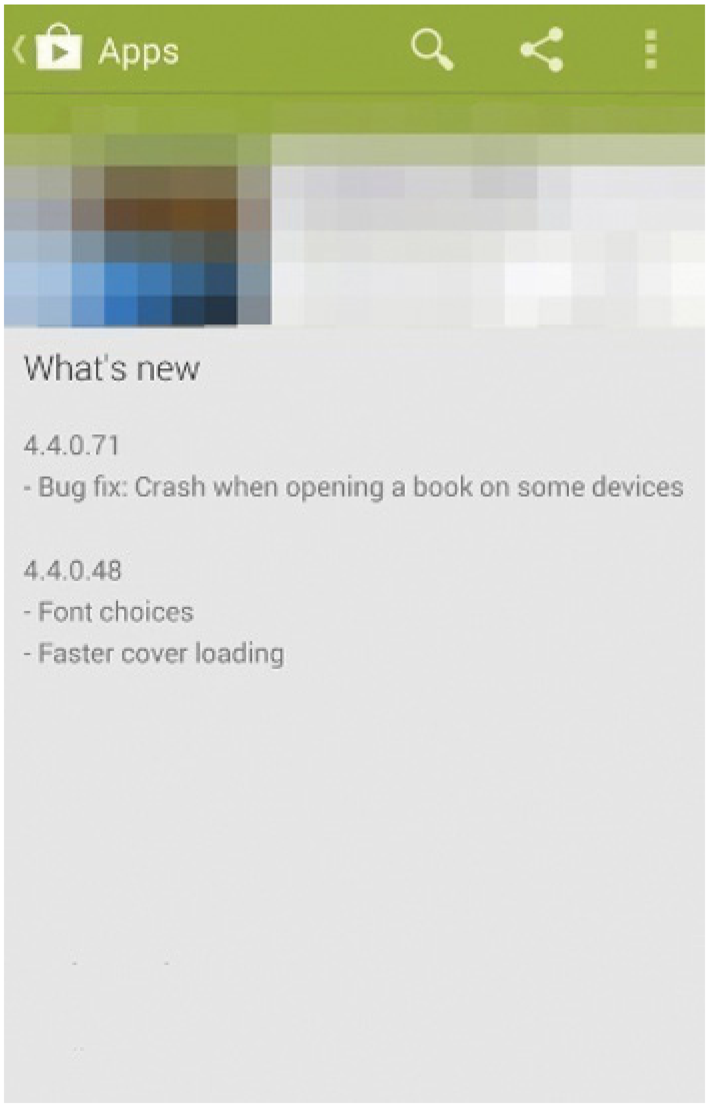


What's new

Fix crashes on some Android 4.3 and x86 devices

Figure 8.1 *App crashes on Android 4.3 and x86 devices*

Nevertheless, as shown in Figure 8.2, some apps have problems only with a certain version on the Android platform, so testing on several mobile operating systems is very important.



What's new

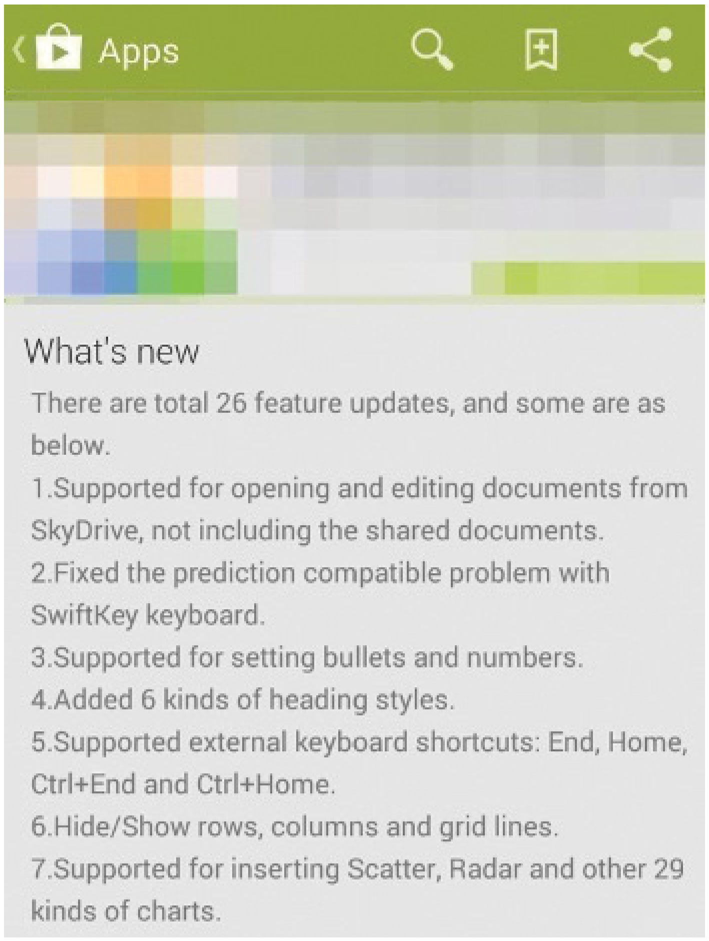
4.4.0.71
- Bug fix: Crash when opening a book on some devices

4.4.0.48
- Font choices
- Faster cover loading

Figure 8.2 *App crashes on some devices*

Keyboards

As I mentioned in Chapter 3, "Challenges in Mobile Testing," users are able to replace system apps such as the keyboard app with a third-party solution. This can lead to various problems as shown in Figure 8.3, which contains a bug report. To reproduce this issue, you need to install the third-party keyboard and start hunting the bug.



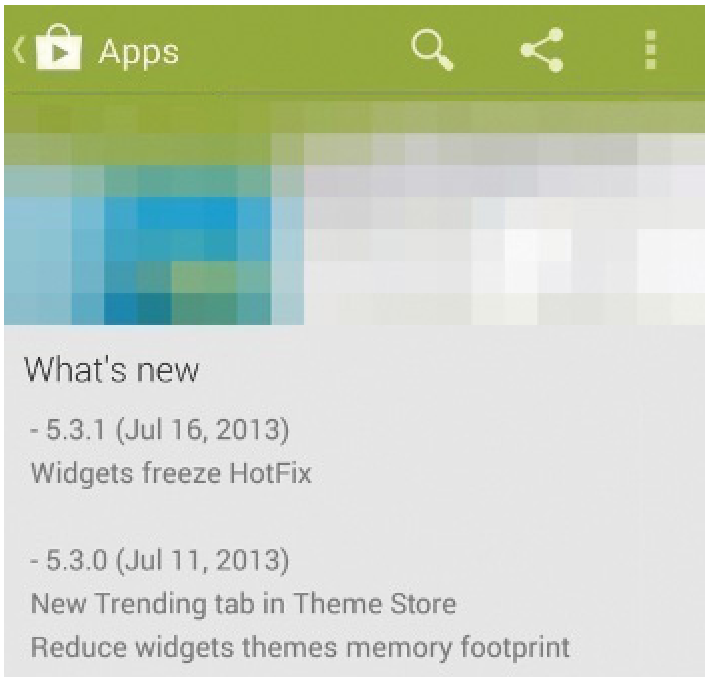
What's new

There are total 26 feature updates, and some are as below.
1.Supported for opening and editing documents from SkyDrive, not including the shared documents.
2.Fixed the prediction compatible problem with SwiftKey keyboard.
3.Supported for setting bullets and numbers.
4.Added 6 kinds of heading styles.
5.Supported external keyboard shortcuts: End, Home, Ctrl+End and Ctrl+Home.
6.Hide/Show rows, columns and grid lines.
7.Supported for inserting Scatter, Radar and other 29 kinds of charts.

Figure 8.3 *App problems due to alternative keyboards*

Widgets

Some mobile platforms support the use of widgets. When providing a widget, be sure that it won’t freeze, crash, or consume too much battery power as shown in Figure 8.4.



What's new

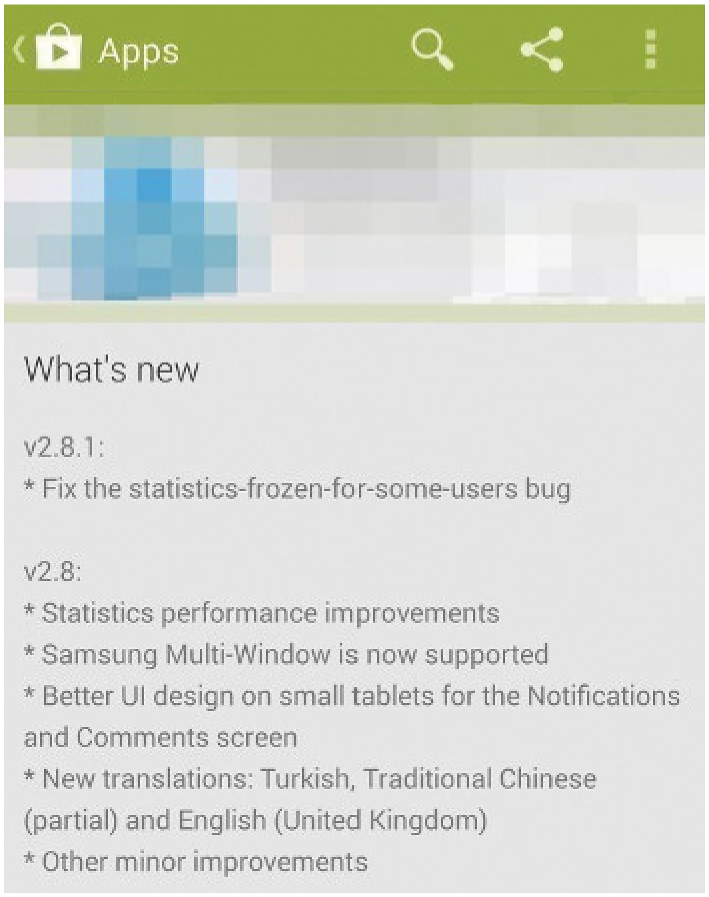
- 5.3.1 (Jul 16, 2013)
Widgets freeze HotFix

- 5.3.0 (Jul 11, 2013)
New Trending tab in Theme Store
Reduce widgets themes memory footprint

Figure 8.4 *Widget consumes too much battery power and freezes.*

Performance

As I have mentioned in several parts of this book, the loading time and performance of an app are essential to a successful app. The app description in Figure 8.5 provides some information about possible performance issues on the statistics page. To reproduce the performance issue of this app you need to get two very similar devices. On the first device you should keep the old version of the app while installing the update on the other device. Then you can compare the performance of the described section as well as the loading times to see if there is an improvement (see Figure 8.6).

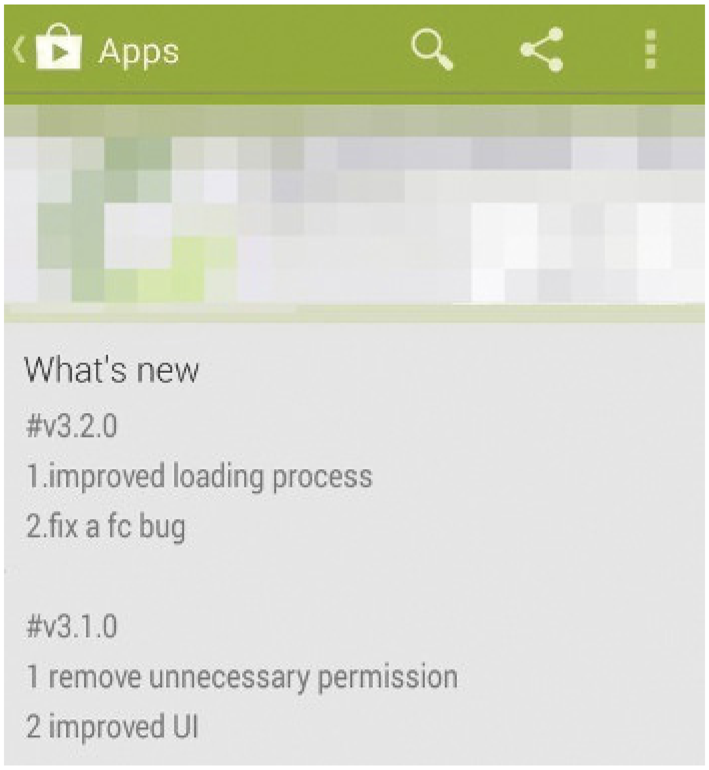


What's new

v2.8.1:
* Fix the statistics-frozen-for-some-users bug

v2.8:
* Statistics performance improvements
* Samsung Multi-Window is now supported
* Better UI design on small tablets for the Notifications and Comments screen
* New translations: Turkish, Traditional Chinese (partial) and English (United Kingdom)
* Other minor improvements

Figure 8.5 *Performance issues in some sections of the app*



What's new

#v3.2.0

1.improved loading process

2.fix a fc bug

#v3.1.0

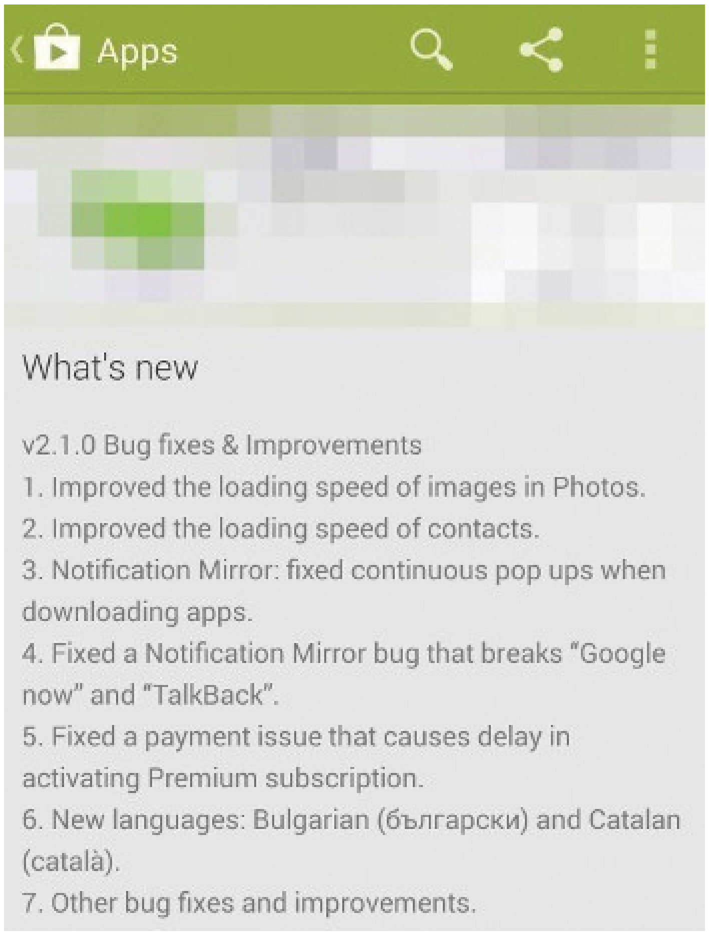
1 remove unnecessary permission

2 improved UI

Figure 8.6 *Loading performance of the app*

Login and Payment

If your app provides a login feature or mobile payment process, it is critical that those features work. If your users can't log in or buy anything, you will lose money and harm your reputation. Features critical to your app must work to the maximum possible extent, so you need to be sure that they're well tested and covered by test automation. Take a look at the screenshot in Figure 8.7, where you can see that the mobile app provider had an issue with its subscription model.



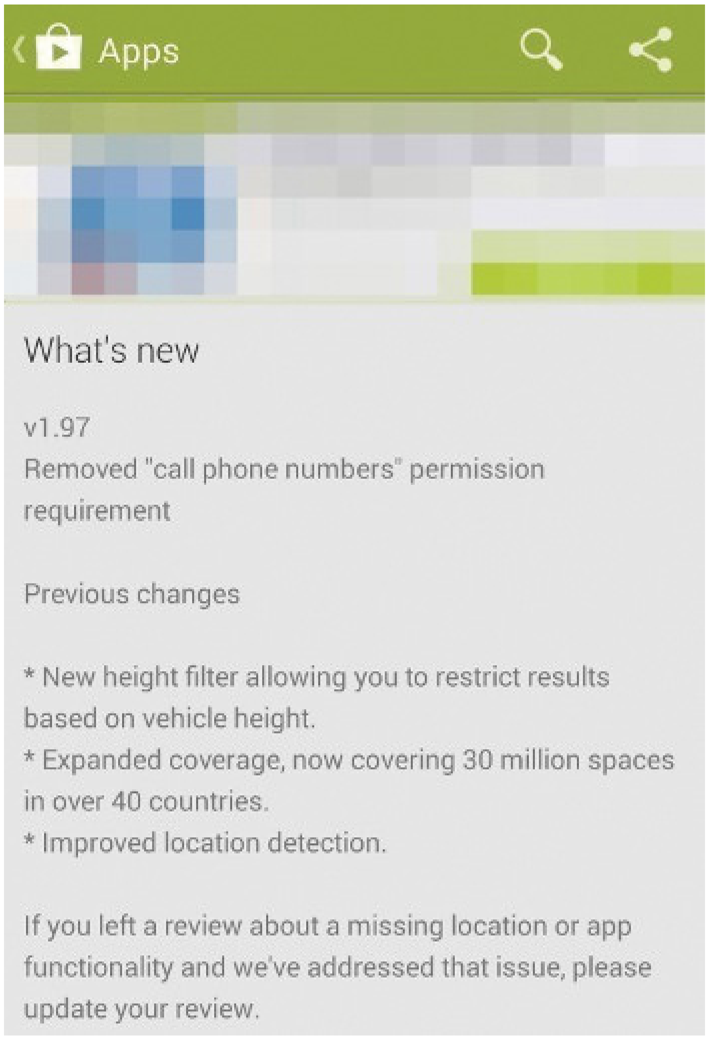
What's new

v2.1.0 Bug fixes & Improvements
1. Improved the loading speed of images in Photos.
2. Improved the loading speed of contacts.
3. Notification Mirror: fixed continuous pop ups when downloading apps.
4. Fixed a Notification Mirror bug that breaks "Google now" and "TalkBack".
5. Fixed a payment issue that causes delay in activating Premium subscription.
6. New languages: Bulgarian (български) and Catalan (català).
7. Other bug fixes and improvements.

Figure 8.7 *Checkout problems with premium subscription*

Permissions

As I mentioned in Chapter 4, "How to Test Mobile Apps," it's very important that you use only the mobile app permissions you really need for your features to work. If you use permissions that users don't understand or that aren't required (see Figure 8.8), you will probably get lots of bad reviews in the stores, and it may even negatively impact your app's security. Be sure to check the permissions again before releasing your app.



What's new

v1.97
Removed "call phone numbers" permission requirement

Previous changes

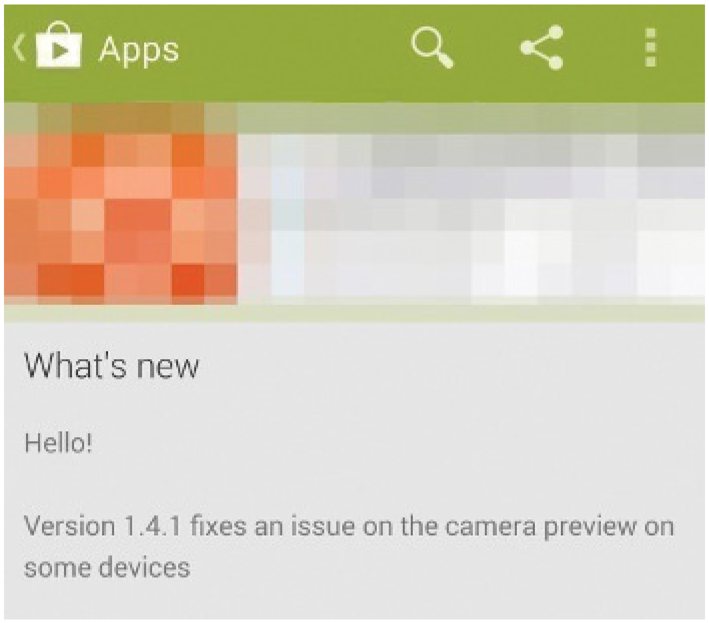
* New height filter allowing you to restrict results based on vehicle height.
* Expanded coverage, now covering 30 million spaces in over 40 countries.
* Improved location detection.

If you left a review about a missing location or app functionality and we've addressed that issue, please update your review.

Figure 8.8 *Using permissions that aren't required*

Mobile Device Hardware Usage

If your app uses mobile-device-specific hardware features such as the camera, be sure that the feature works on as many devices as possible. As you can see in Figure 8.9, the app provider had an issue with the camera preview on some devices. Testing for hardware support on different devices is a good task for crowd testers.



Hello!

Version 1.4.1 fixes an issue on the camera preview on some devices

Figure 8.9 *Camera preview not working properly*

Since coming up with the idea of checking app update texts, I have continued doing this every time before I update to a newer version of an app. This is sometimes time-consuming and frustrating when I'm unable to reproduce the described bug, but it helped me a lot in improving my mobile testing skills. Just by reading through the app descriptions I learned so many new ways of generating mobile testing ideas as well as lots of new ways to provoke bugs during my daily testing work.

Observe

Another way to improve your mobile skills is by observing other people. Watch other people while they're using their mobile devices. Try to observe other people when among the public, for example, on a train, in the supermarket, or anywhere they're using an app. It is very interesting to see how other people use apps in totally different ways, and you can learn all sorts of things from your observations that you can reuse while testing your app and planning new features with your team. It will also help you to generate new testing ideas and bear other usage behaviors in mind.

Important

Don't be too obvious and don't start stalking people while observing them.

During my observations I have noticed that many people don't know that a navigation drawer can be opened by swiping from the left of the screen to the right. They touch the navigation icon in the top left-hand corner to open it. I realized that not everyone is familiar with every feature that a mobile device or app has to offer. You should therefore take the time to observe other people in the wild or in test labs to see how they use their apps and then apply this knowledge in your mobile development process.

As a starting point I recommend that you observe either your colleagues or your family. This way you'll learn a lot, and they probably won't mind you observing them.

Take Part in Competitions and Test Cycles

If you're interested in competing with other software testers from around the world and learning from their testing experience, I recommend that you take part in a testing competition. There are lots of competitions throughout the year where software testers can register either as a team or alone to test a piece of software in different categories. The great thing about competitions is that you can share your knowledge and learn from other software testers. It's also fun to compete with other software testers to see how good your testing skills are compared to theirs.

I generally attend testing competitions in order to learn. I don't care about my final competition ranking; I just want to learn and improve my testing skills. I really like to see other testers during their work and pick up new testing ideas.

Here are a couple of testing competitions:

- Software Testing World Cup (www.softwaretestingworldcup.com/)
- Testathon (http://testathon.co/)

Another nice way to improve your testing skills and contribute to the testing community is Weekend Testing.[6] Weekend Testing is a platform where software testers can collaborate and learn from the testing community. As the name suggests, Weekend Testing takes place on the weekend and is all about testing software from different perspectives and sharing your testing work with other software testers. Have a look at the Weekend Testing Web site to check out the upcoming testing dates.

6. http://weekendtesting.com/

As mentioned in Chapter 6, "Additional Mobile Testing Methods," crowd testing is a software testing approach where you add software testers from all over the world to your in-house testing efforts. However, crowd testing can serve another purpose as well: Registering as a crowd tester and attending testing cycles is a great way to learn as you get to see other mobile apps, the problems the app provider faces, and the bugs that come up. Furthermore, it is very interesting to see if your bug reports are good enough to be accepted by the crowd testing provider and mobile app provider.

Important

Take part in testing competitions, share your knowledge with other software testers, and register with crowd testing platforms to see how other mobile apps work. Remember: While it may be interesting to see if your bug reports are accepted by the provider, your focus should be on improving your mobile testing skills.

The Mobile Community and the Mobile World

As mentioned in the previous section, learning from other software testers and mobile testers is a great way to improve your own mobile testing skills. I therefore recommend that you become an active part of the mobile community, for example, by registering on software testing platforms like Software Testing Club.[7]

[7]. www.softwaretestingclub.com/

Every software testing platform also has a mobile section where you can exchange news and views with other mobile testers. It's also very useful to join mobile testing groups on various social media platforms. There are always lots of great mobile developers and testers who want to interact on a certain topic or who have a problem. Helping someone with a problem is a great way to contribute to the mobile community. And don't be shy about asking questions even if you think they might be silly. There's no such thing as a stupid question!

If you don't have a Twitter account yet, I highly recommend that you create one. Every software testing and mobile testing expert uses Twitter to write about new testing ideas, blog posts, and other important information that is bound to help you during your daily working life. Most mobile testing experts also blog, so you should subscribe to their updates to get the latest information from them. In the next section I will list a few blogs and books that may be of interest to you.

Testing events and conferences are another way of exchanging with the mobile community. Lots of software testing conferences are hosted all over the world where software testing and mobile testing experts meet and share their knowledge in the form of talks or workshops. If you have the opportunity to do so, I highly recommend that you attend some conferences and meet other mobile testers in person to talk about a range of topics. You can also check to see if there are any testing user groups in your area; these are generally free to attend and a great opportunity to meet other software testers nearby.

If you work in the mobile testing business, you could start blogging to keep a record of your experience and contribute to the mobile community. You can help other mobile testers with your acquired knowledge or blog about your experiences while becoming a mobile tester.

Besides learning from other mobile testers, it's very important that you stay up-to-date with regard to the latest technologies and features of the mobile operating systems and devices. You need to know when the different mobile device manufacturers release new devices and new versions of their operating systems. For a complete overview of such features, I recommend that you watch the keynote videos posted by the major manufacturers.

I also recommend that you use as many apps as possible from a range of different categories so you always have an overview of possible new features as well as new ways to implement and use an app.

Valuable Sources

This section of the chapter provides you with some interesting software testing communities, books, magazines, and blogs that you can use to improve your knowledge.

Important

These lists are by no means complete.

Conferences

The following conferences are worth going to:

- Agile Testing Days (www.agiletestingdays.com/)
- Belgium Testing Days (http://btdconf.com/)
- Dutch Testing Day (www.testdag.nl/)
- EuroSTAR (www.eurostarconferences.com/)
- Google Test Automation Conference (https://developers.google.com/google-test-automation-conference/)
- Iqnite (www.iqnite-conferences.com/index.aspx)
- Let's Test (http://lets-test.com/)
- Mobile App Europe (http://mobileappeurope.com/)
- Øredev (http://oredev.org/)
- STAREAST (http://stareast.techwell.com/)
- STARWEST (http://starwest.techwell.com/)
- TestBash (www.ministryoftesting.com/training-events/testbash/)
- TestExpo (http://testexpo.co.uk/)

Communities

The following software testing communities are worth looking into:

- Association for Software Testing (www.associationforsoftwaretesting.org/)
- Mobile QA Zone (www.mobileqazone.com/)
- Software Testing Club (www.softwaretestingclub.com)
- Testing Circus (www.testingcircus.com/)
- uTest community (www.utest.com)

Books

The following books are worth reading. Not all of them are about mobile testing, but they are an excellent source of knowledge for software testers.

- *Agile Testing* (http://lisacrispin.com/agile-testing-book-is-now-out/) by Lisa Crispin and Janet Gregory

- *Beautiful Testing* (www.amazon.com/gp/product/0596159811?tag=sw-testing-books-20) edited by Tim Riley and Adam Goucher
- *Explore It!* (http://pragprog.com/book/ehxta/explore-it) by Elisabeth Hendrickson
- *How Google Tests Software* (http://books.google.de/books?id=VrAx1ATf-RoC) by James A. Whittaker, Jason Arbon, and Jeff Carollo
- *Lessons Learned in Software Testing* (www.amazon.com/gp/product/0471081124?tag=sw-testing-books-20) by Cem Kaner, James Bach, and Bret Pettichord
- *Specification by Example* (http://specificationbyexample.com/) by Gojko Adzic
- *Tap into Mobile Application Testing* (https://leanpub.com/testmobileapps) by Jonathan Kohl

Magazines

The following magazines provide great content from experts in various industries. These magazines focus on specific software and mobile testing topics.

- *Agile Record* (www.agilerecord.com/)
- *Professional Tester* (www.professionaltester.com/magazine/)
- *Tea-time with Testers* (www.teatimewithtesters.com/)
- *Testing Circus* (www.testingcircus.com/)

Blogs

The following list contains great blogs from software testing experts:

- Gojko Adzic, http://gojko.net/
- James Bach, www.satisfice.com/blog/
- Michael Bolton, www.developsense.com/blog/
- Lisa Crispin, http://lisacrispin.com/
- Martin Fowler, http://martinfowler.com/
- Markus Gärtner, http://blog.shino.de/
- Shmuel Gershon, http://testing.gershon.info/

- Andy Glover, http://cartoontester.blogspot.co.uk/
- Adam Goucher, http://adam.goucher.ca/
- Elisabeth Hendrickson, http://testobsessed.com/
- Jim Holmes, http://frazzleddad.blogspot.com/
- Lena Houser, http://trancecyberiantester.blogspot.com/
- Eric Jacobson, www.testthisblog.com/
- Stephen Janaway, http://stephenjanaway.co.uk/stephenjanaway/blog/
- Viktor Johansson, http://therollingtester.com/
- Jonathan Kohl, www.kohl.ca/blog/
- Rob Lambert, http://thesocialtester.co.uk/
- Alan Page, http://angryweasel.com/blog/
- Huib Schoots, www.huibschoots.nl/wordpress/
- Rosie Sherry, www.rosiesherry.com/

The following blogs have multiple contributors:

- http://blog.inthewildtesting.com/
- http://blog.utest.com/ (uTest employees)
- http://googletesting.blogspot.de/ (Google employees)
- www.ministryoftesting.com/testing-feeds/ (a great testing feed collection)
- http://mobileapptesting.com/
- http://webapptesting.com/

And then there’s my blog:

- www.adventuresinqa.com

Summary

Chapter 8 was all about skills and the required skill set of a mobile tester. Hiring mobile testers is not easy, because they are very rare. If you have found someone who could fit your company and position, be sure he or she has the following soft skills in order to be successful in the role of mobile tester:

- Communication

- Curiosity
- Critical thinking
- Tenacity
- Constant learner
- Creativity
- Customer focus
- Programming and technical skills

If you are a mobile tester and want to improve your testing skills, this chapter provided some suggestions. With lots of examples, the section "Learn from Other Apps" showed you how to improve your testing skills by trying to reproduce bugs in existing apps. In addition, it is important to be active in the mobile testing community, to learn from other mobile testers and to share your knowledge. At the end of the chapter an overview of important conferences, books, blogs, and magazines was provided.

Chapter 9. What's Next? And Final Thoughts

Welcome to the final chapter of this book! This chapter deals with the question "What's next?"

What's the next big thing mobile testers are going to have to deal with? What kinds of new technology are already out there, are on their way, or could arrive in the future? Are there any new testing challenges that we need to handle? Are there any new testing tools and test environments on the horizon?

To answer those questions, the following sections of this chapter describe some new technologies that are already on the market or due to arrive in the near future. To keep pace in the fast-growing world of technology, it's important for you to know what's in the pipeline.

The following sections can be used for further investigations and research if needed. No one is able to predict the future, but I'm convinced that the following technologies will become more and more important for mobile testers in the next couple of years.

Internet of Things

The Internet of Things (IoT) refers to the interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure to offer different kinds of services. Things in the IoT can include a wide variety of devices such as human medical implants, biochips for animals, and cars with built-in sensors that communicate with one another to exchange information about the current traffic situation or provide drivers with certain information about their cars. There are also devices such as washing or coffee machines that can connect to the Internet so you can monitor them remotely. Everything that can be assigned an IP address and has the ability to provide and transfer data over a network is a thing in the IoT.

According to a study conducted by Gartner,[1] there will be up to 26 billion devices on the Internet of Things by 2020. And that figure doesn't include computers, tablets, or smartphones, which will reach the 7.3 billion mark by 2020. If you compare those numbers, IoT devices will dwarf all the existing smart devices. This will of course give rise to a whole new industry with people trying to connect everything to the Internet.

1. www.gartner.com/newsroom/id/2636073

Here are some possible usage areas and scenarios for IoT:

- **Environmental monitoring:** Sensors can be used to monitor water quality, soil conditions, or the atmosphere.
- **Infrastructure management:** Bridges, railway tracks, and wind farms can be monitored.
- **Energy management:** Industrial manufacturers can optimize energy levels in real time.
- **Medical and health care systems:** People's health can be monitored remotely.
- **Building and home automation:** Alarm and heating systems can be monitored and managed.
- **Transport systems:** Cars can communicate with one another, such as to avoid traffic.

In order to standardize the IoT, a consortium[2] of companies has been formed to push along the IoT and develop default communication strategies, interfaces, and protocols. The following two sections provide some examples of current IoT devices and scenarios from different manufacturers.

2. www.openinterconnect.org/

Connected Home

Mobile device manufacturers Google and Apple are currently building their first IoT services and products which will form part of the IoT family alongside several other companies. In 2014, Google bought Nest Labs,[3] which builds intelligent thermostats and smoke alarms for smart homes. The thermostats and smoke alarms are connected to a Wi-Fi network and can be accessed from anywhere in the world using a computer, tablet, or smartphone.

3. https://nest.com/

Google is currently developing mobile apps for different mobile platforms that grant access to connected devices around the home. The product is intelligent as it can learn from user habits to control the heating system based on the time of day and the current month and determine whether or not the user is at home. Users can also define various scenarios to control their entire home heating system based on their needs. Because the devices are connected to the Internet, it's easy to auto-update them with new software versions including bug fixes and features.

Apple has introduced HomeKit[4] with iOS 8, which provides a framework to communicate and control connected devices in the user's home. Users will be able to automate and control IoT devices with their voice by using Siri. HomeKit will be able to control devices such as thermostats, lighting, doors, and other elements that can connect to the Internet. Apple provides the development framework for HomeKit and is currently looking for industry partners who want to implement Apple's HomeKit accessory protocol to be controlled by the HomeKit app.

4. https://developer.apple.com/homekit/

As you can see, connected homes are already available on the market and enable entirely new ways of interacting with devices and parts of our daily life. There are plenty of potential new test scenarios, test environments, and testing challenges that are totally different from traditional testing or mobile testing.

Important

Besides Google and Apple, there are a lot of other companies that are investing in and already have solutions in place for connected homes. I opted to write about Google and Apple because they also provide APIs for developers that allow them to build mobile applications around the connected home technology.

Connected Car

Connected cars are the next IoT example. Again, Google and Apple are already on their way to integrating the Android and iOS mobile operating systems into cars to make them even more intelligent. Google has introduced Android Auto,[5] and Apple released CarPlay.[6] Both Google and Apple will provide a lightweight version of their mobile operating systems to provide users with features they can use while driving, such as navigation, music, contacts, phone, and messages.

5. www.android.com/auto/
6. www.apple.com/ios/carplay/

Besides their own mobile apps, both Google and Apple offer the option to use installed third-party apps in cars. There are already lots of car manufacturers that support both systems and let buyers choose the system they prefer.

However, using mobile apps with car displays gives rise to some new challenges for anyone involved in the software development process. For example, the provided apps and features shouldn't distract the driver and need to offer a very simple user interface with less information compared to mobile apps or Web applications.

The following four points must be considered when developing and testing mobile apps or other applications for connected cars:

- **Simple interface:** Car applications and interfaces should not distract the driver. The UI elements must be easy to use while the driver is at the wheel. Traditional input methods need to be reconsidered and should also include voice control.
- **Avoid useless features:** The feature set of an app required while driving is probably a lot smaller than for an app used on a smartphone. The app should therefore offer fewer features on the car display to prevent the driver from being overwhelmed or frustrated while at the wheel.
- **Third-party apps need guidelines:** Car manufacturers need to provide an API for third-party developers so their services can be integrated. However, this poses the challenge of creating very strict guidelines about what is and isn't possible.

- **Testing in the car:** Developing apps for cars is challenging enough in itself, but testing apps for cars is even more complex. It is simply not enough to test the app in a lab situation because cars are generally on the move, have different manufacturer years and models, and have lots of interfaces with other systems. Electronic interference in a car can have a huge impact on your app and the whole system. The provided app must be safe for the driver to use while at the wheel. Last but not least, the app must be well tested to avoid any critical bugs.

A good example of a connected car and IoT is the car manufacturer Tesla,[7] which builds cars that are completely connected to the Internet and can be partially controlled with the aid of a mobile app. The car receives automatic updates that improve features, fix bugs, and even solve problems with different parts of the engine. *Wired* magazine published an interesting article about Tesla as an example of IoT.[8]

7. www.teslamotors.com/

8. www.wired.com/2014/02/teslas-air-fix-best-example-yet-internet-things/

As you can see, the two examples—connected home and car—represent new challenges for the entire software development process. From a testing point of view in particular, these new technologies require different testing methods as well as new testing environments, new testing devices, and completely new testing scenarios.

Wearables

Wearable technology is a rapidly growing field that's expected to grow exponentially over the next few years. There are lots of new and innovative form factors for devices that can be worn on different parts of the body. These new form factors pose new challenges for companies as they look to find smart ways to make their product functional, usable, and lovable for their customers. The same applies to developers and software testers who need to rethink their work and the way they develop and test such products. Wearables generally involve smart watches, smart glasses, and fitness wristbands.

Smart Watches and Fitness Wristbands

Smart watches and fitness wristbands are extensions to mobile devices that send and receive information such as messages, news, incoming calls, and health status to and from mobile devices. To get the latest information from an app, users no longer need to take their device out of their pocket. Most devices can be controlled by the user's voice or a small touchscreen. However, a smart watch or fitness wristband is essentially useless without a mobile device to interact with.

The usability and design of smart watches and fitness wristbands need to be thoroughly tested and checked. Designing software for really small screens is not easy, which is why designers and UX experts need to rethink their concepts in order to build nice products that wearables users will love and use. Jonathan Kohl wrote an excellent article about his lessons learned when designing products for smart watches and wearables.[9]

9. www.kohl.ca/2014/lessons-learned-when-designing-products-for-smartwatches-wearables/

If you have the opportunity to test wearables, especially smart watches or fitness wristbands, you should keep an eye on the look-and-feel of the device together with the software under test. This also includes testing the design and usability. When doing so, you should ask yourself the following questions to generate valuable feedback about the product:

- Is the device nice to wear?
- Does the app make sense on the wearable device?
- Are the features easy to use and helpful?
- Do certain parts of the device get in the way while you're on the move and while you're using the software?
- How can the user interact with the smart watch or fitness wristband?

The look, feel, design, and usability are the main success factors when it comes to wearable technology. If a wearable device doesn't feel good, users will not buy or wear it.

From a technological point of view, testing smart watches has some additional challenges compared to mobile apps and devices. The fact that smart watches are extensions to mobile devices requires testing of the wearable device together with the software to see how both communicate with mobile devices in order to receive and transfer data. This scenario isn't something you can automate. I'm sure you're already aware of mobile device fragmentation, but this is compounded by the fact that smart watches and fitness wristbands need to work correctly within a different set of unique daily user scenarios, and all those scenarios require extensive in-the-wild testing. Such testing in a real-life environment will play an essential part in the success of software for smart watches.

Google introduced Android Wear[10] in 2014 to kick off the wearable device era. Apple introduced the Apple Watch[11] in September 2014 and started selling it in early 2015. If you search the Internet for smart watches and fitness wristbands, you'll come across various device manufacturers and all the different devices on the market.

10. www.android.com/wear/

11. www.apple.com/watch/

For information about building software products for wearable devices, check out the Pebble developer,[12] the Google wearable,[13] and the Apple Watch[14] feature pages.

12. https://developer.getpebble.com/

13. https://developer.android.com/training/building-wearables.html

14. www.apple.com/watch/features/

Smart Glasses

Google Glass is another wearable that was introduced by Google. Google Glass[15] includes almost the same hardware as a mobile device extended with an optical head-mounted display (OHMD) to have the content and information directly in front of your eyes. The glasses are equipped with lots of sensors and a camera to interact with your surroundings. You can control this wearable device with your voice or by using the touchpad on the side of the frame.

15. www.google.com/glass/start/

The explorer edition of Google Glass has been on sale in some countries since 2014, but lots of countries and companies have expressed privacy concerns since its introduction in 2012 because the device is able to record people in public without their permission. Furthermore, there are concerns about the product in terms of corporate secrets and safety considerations while using it in different scenarios, such as while driving a car or riding a motorbike.

However, Google provided a new way of using mobile technologies and set new standards and innovations in the world of wearable devices, even though the product isn't ready for the mass market yet. Google Glass is a great example of the direction technology will take over the coming years.

If you have the opportunity to develop and test software for smart glasses, don't forget to run through the list of questions applicable to smart watches. You'll also need to rethink your testing approach for this device.

Health Apps

Another interesting and growing market is mobile health apps. "The number of mHealth apps published on the two leading platforms, iOS and Android, have more than doubled in only 2.5 years to reach more than 100,000 apps (Q1 2014)" and ". . . will reach $26 billion in revenues by 2017, . . ." as quoted in the current mHealth App Developer Economics[16] report. This huge increase shows that mHealth apps will be on the rise in the near future.

16. http://mhealtheconomics.com/mhealth-developer-economics-report/

The top four mHealth apps are:

1. **Fitness** apps (30%)
2. **Medical reference** apps (16%)
3. **Well-being** apps (15%)
4. **Nutrition** apps (8%)

The remainder of the mHealth apps are distributed among different categories such as medical condition management, diagnostics, compliance, reminders, alerts, and monitoring.

Health apps involve the use of mobile devices or wearables to monitor the human body for current blood pressure, pulse, heart rate, sleep patterns, calorie consumption, or current speed while running. The huge number of mHealth apps on the two leading platforms, iOS and Android, has convinced Apple and Google to invest in and develop mHealth APIs and apps for their mobile platforms. Apple introduced Health[17] for its customers and HealthKit[18] for its developers at the same time as it rolled out iOS 8. Google introduced Google Fit[19] in 2014. The fact that both Apple and Google are entering the health market will lead to several new devices and apps being rolled out in the near future.

17. www.apple.com/ios/whats-new/health/

18. https://developer.apple.com/healthkit/

19. https://developers.google.com/fit/

As the numbers show, most of these apps help customers to track their fitness or dietary habits, but health apps also pose high risks to customers. Apps that manage insulin doses for patients with diabetes could have disastrous consequences if a bug occurs. This risk raises an important question: Can we trust health care apps?

Medical devices are generally regulated by the United States Food and Drug Administration (FDA), but this is not the case with every mobile health app. Experts from the *New England Journal of Medicine*[20] say that the FDA doesn't have enough resources to regulate all the health apps available in the different mobile app stores. Another challenge that is nearly impossible for the FDA and app providers to handle is all the mobile operating system updates provided by the different vendors. Each mobile platform receives more than one or two updates a year, and each operating system update must be compliant with the FDA regulations.

20. www.nejm.org/doi/full/10.1056/NEJMhle1403384

So the answer to the previous question is **no**. We can't trust health apps if they aren't regulated by any institution because we can't be sure that the delivered data is correct and free of mistakes.

If you have the opportunity to test mobile health and fitness apps, please bear the following points in mind:

- Get information from the FDA and other medical institutions with regard to regulations and health care workflows.
- The provided data *must* be correct in order to protect human life.
- Data security is a very important aspect due to the privacy of a person's state of health.
- Mobile health apps must have excellent usability in order to cover the target group's needs.
- Geolocation data must be correct for fitness trackers.

Besides that, all of the mobile testing knowledge you have acquired in this book also applies to health and fitness apps.

If you'd like to find out more about mHealth, visit the *mHealthNews*[21] or mobile Health Economics[22] Web sites or have a look at the mHealth App Developer Economics 2014 study.[23]

21. www.mhealthnews.com/
22. http://mhealtheconomics.com/
23. http://mhealtheconomics.com/mhealth-developer-economics-report/

Final Thoughts

This is the last section of my book about mobile testing, and I'd like to provide you with some final thoughts. During the course of this book you've learned a great deal about mobile devices, mobile apps, mobile users, and the tools that are important when it comes to mobile testing. I also hope you've learned from my ideas and experiences during my time as a mobile tester.

This book is designed to help you in your daily life as a mobile tester, mobile developer, or product manager by giving you the impetus to generate new testing ideas and try out new mobile testing approaches. It should also serve as a basis for developing your own testing ideas and approaches, while also helping you to extend your knowledge level.

As you have seen, the mobile development and testing business is rapidly changing with lots of new technology entering the market every day and plenty more in the pipeline. This is why it's important that you stay up-to-date, strive to learn constantly, and adapt your skills to the ever-changing world of technology.

Five Key Success Factors

To round things off, I'd like to provide you with my five key success factors for becoming a successful mobile tester.

Success Factor 1: Have High Expectations

Mobile users have high expectations, and you should also have very high expectations when it comes to mobile apps and their usability, performance, and feature set. Bear in mind that mobile customers will uninstall your app very quickly if they're not happy with it, and they'll probably submit a bad review to the app store. It's therefore important that you always keep your customers in mind, encourage good usability and performance, and make sure that all the important bugs get fixed. Keep the KIFSU principle in mind and listen to your customers' needs.

Success Factor 2: Be an Expert on Mobile Devices

A successful mobile tester needs to be an expert on mobile devices. It is essential to know all the different hardware and software features of mobile devices from the various platforms. This knowledge will help you to keep lots of different test scenarios in mind during your daily business. If you're able to do so, buy mobile devices using the different platforms so you can stay up-to-date. If buying is not an option, try to rent them from a mobile device lab.

You should also subscribe to various technology blogs and news pages in order to get the latest news about mobile operating systems and mobile devices. I recommend that you watch the keynote videos of the major mobile manufacturers to find out the latest information about the different platforms.

Success Factor 3: Be on the Move

One of the most important points you should bear in mind when testing a mobile app is to be on the move while you're testing. Your customers use their mobile apps in many different scenarios, locations, and data networks. Therefore, it's essential that you test your app in several data networks with different network speeds to replicate real-life scenarios. While testing your app in the wild, you'll doubtless come across lots of different problems that would probably never show up in the office. When testing an app on the move, there will be lots of interferences that could have an impact on your app when using the various sensors and interfaces a mobile device has on offer.

So grab a bag, fill it with mobile devices, and start testing in the wild right now!

Success Factor 4: Build Up Your Programming Skills

Mobile testers need to be able to write test automation code. If you don't have any programming skills right now, do your best to get into programming so you can write reliable and robust test automation scripts for your mobile app. Programming skills will also help you support the developers with their regression tests, and you'll be able to communicate and discuss the app's code with the developers. If your programming skills need brushing up, now's the time to read some programming language books or run through some online tutorials.

Success Factor 5: Be a Constant Learner

The final success factor in becoming a better mobile tester is to be a constant learner. This doesn't just apply to mobile testers; it should be the case for anyone involved in the IT business. The technologies used to build complex systems including mobile apps are changing constantly. Furthermore, new ways of using and communicating with new technologies are on their way, and it's important for you to find out about them as soon as possible.

Besides learning new technologies, you should also work on improving your testing skills. Great ways to do this include reading lots of blogs and books, attending conferences, and taking part in competitions to learn from other mobile testers and share experiences with them. This will help you to improve your testing ideas, approaches, and skills.

Don't shy away from trying out new things—make mistakes and learn from them.

Summary

The last chapter of this book covered the topic "What's next?" What are the upcoming technology trends software testers have to deal with? I described five possible technology trends that are already on the market or on their way. The five technologies are

- Internet of Things
- Connected homes
- Connected cars
- Wearables
- Health apps

In the “Final Thoughts” section I outlined five key success factors to become a successful mobile tester. Those success factors are

- Have high expectations.
- Be an expert on mobile devices.
- Be on the move.
- Build up your programming skills.
- Be a constant learner.

That’s it; thank you very much for reading my book. I hope you learned a lot of new things and got new ideas for your daily life as a mobile tester.

Happy mobile testing!

Index

0G networks, 12
1G networks, 12
2G networks, 12–13
3G networks, 13
4G networks, 13
5G mobile standard, 13
99tests, 145

Acceleration sensor, testing, 38
Accessibility guidelines
    Android, 62
    iOS, 62
    WC3 initiative, 62
    Windows Phone, 63
Accessibility testing
    Android, 62
    auditory impairment, 61
    cognitive impairment, 61–62
    iOS, 62
    overview, 60
    physical impairment, 61
    visual impairment, 60–61
    WC3 initiative, 62
    Windows Phone, 63
Actual result, in bug reports, 98
Advanced User Interactions API, 125
Advertisements, selling, 24
Agile Record, 197
Agile Testing, 196
Agile Testing Days, 195
Amazon Appstore Guidelines, 26

Ambient light sensor, testing, 37
AMPS (Advanced Mobile Phone Service), 12
AMTS (Advanced Mobile Telephone Service), 12
Android
  design guide, 18
  guidelines, 81
Android apps
  test automation. *See* Test automation tools, Android.
  testing criteria, 103
Android Asset Packaging Tool, 66
Android Auto, 201
Android Emulator, 51
Android operating system
  accessibility guidelines, 62
  accessibility testing, 62
  App Quality Alliance site, 103
  pseudo-random test events, 66
Android smartphone
  battery status, displaying, 18
  dock section, 18
  generating pseudo-random test events, 66
  interrupt testing, 66–67
  introduction of, 14
  vs. iOS, 17–19
  JouleUnit energy profiling tool, 65
  Monkey tool, 66
  network strength, displaying, 18
  notifications, displaying, 18
  pinning apps, 18
  status bar, 18
  stress testing, 66–67
  time, displaying, 18
  user interface, 18

Android Web browser *vs.* other browsers, 48
App development
  development guidelines, 2–3
  KIFSU (Keep It Fast Simple Usable) principle, 3
  KISS (Keep It Simple, Stupid) principle, 3
  rejection, reasons for, 18
  testing. *See* Mobile testing.
App Quality Alliance site, 103
App stores
  Apple App Store, 25
  BlackBerry World, 26
  Google Play Store, 25
  list of, 25
  Microsoft Store, 26
  number of available apps, 25
  rejecting apps, reasons for, 18
  review and publishing guidelines, 26
  Samsung, 25
  uploading apps to, 26
  for Windows Phone, 26
Appaloosa, 138
AppBlade, 138
Appium
  for Android, 127–128
  for iOS, 134
  native object recognition, 111
  online resources, 111, 134
Applause, 138, 145
Apple App Store, 25
Apple App Store Guidelines, 26
Apple Watch, 204
"Applying the SFDPOT Heuristic to Mobile Testing," 88
Apps

developing. *See* App development.
health, 205–207
mHealth, 205–207
number available, 2, 25
pinning, 18
preinstalled, testing, 44
reasons for deleting, 2
system, testing, 44
testing. *See* Mobile testing.
uploading to app stores, 26
Windows Phone, downloading, 26
Apps, Android
test automation. *See* Test automation tools, Android.
testing criteria, 103
Apps, iOS
test automation. *See* Test automation tools, iOS.
testing criteria, 103
Apps, types of
hybrid, 20–21
native, 19–20
Web, 22–23
appsfire, 177
AppsFlyer, 177
AppThwack, 36, 149
Appurify, 149
Association for Software Testing, 196
Auditory impairment, accessibility testing, 61
Automated testing. *See* Test automation.

Bach, James, 88, 90
Bamboo, 137
Battery consumption tools, 64–66
Battery life, saving with proximity sensor, 37–38
Battery status, displaying, 18

Battery usage, testing
battery consumption tools, 64–66
Charles tool, 63
Fiddler tool, 64
fully charged battery, 63–64
Instruments energy usage function, 65
JouleUnit energy profiling tool, 65
low battery, 64
overheating, 64
usage statistics, 64–66
Beautiful Testing, 196
Belgium Testing Days, 195
Beta by Crashlytics, 138
Beta distribution tools, 138–139
BirdFlight, 138
Black Box Software Testing, 55
Black box testing, 54–55, 164
BlackBerry
guidelines, 81
relative market share, 14, 16
BlackBerry Web browser *vs.* other browsers, 48
BlackBerry World, 26
BlackBerry World Guidelines, 26
Blindness. *See* Visual impairment.
Blink, 47
Blogs, 196–197
Books about SQLite, 75
Books and publications. *See also* Online resources.
Agile Record, 197
Agile Testing, 196
Beautiful Testing, 196
Explore It!, 196
How Google Tests Software, 196

The Inmates Are Running the Asylum, 30
Lessons Learned in Software Testing, 196
Mobile Usability, 57
Professional Tester, 197
Specification by Example, 196
Tap into Mobile Application Testing, 92, 196
Tea-time with Testers, 197
Testing Circus, 197
Bots, 137
Boundary value testing, 55
Browser version, in bug reports, 99
Bug Advocacy, 183
Bug category, in bug reports, 101
Bug ID, in bug reports, 97
Bug reports
be polite, 102
keep it simple, 102
mind maps, examples, 94–96
overview, 93, 96–97
remain calm, 102
Bug reports, contents
actual result, 98
browser version, 99
bug category, 101
bug ID, 97
description, 97
expected result, 98
language, 100
log files, 101–102
mobile device, 99
mobile platform, 99
mobile-device-specific information, 99
network condition and environment, 100

operating system, 99
reproducible, 99
screenshot or video, 101
severity, 101
software build version, 100
steps to reproduce, 98
test data, 100
tester who found the bug, 102
work-around, 98
Bugs, finding and protecting from, 53–55
Buildbot, 137
Business models
choosing, 24–25
freemium, 23–24
in-app purchases, 24
paid, 24
selling advertisements, 24
transaction, 24

Calabash
for Android, 125–126
for iOS, 131–132
native object recognition, 111
Cameras
mobile testing, 43–44
resolution and image size, 43–44
testing, 43–44
Capacitive touchscreens, 41
Capture and replay tools, 111–112
CarPlay, 201
CDMA (Code Division Multiple Access), 12
Cell phones
generations of, 12–13
heads, 12

history of, 4–5
transceivers, 12
Charles battery tool, 63
Checklists. *See also* Guidelines; Mind maps; Mnemonics.
generic mobile testing, 85–87
mobile test strategy, 169–170
overview, 84
Checkstyle tool, 54
Chrome Web browser *vs.* other browsers, 48
CI systems, 137
Clicktale, 177
Cloud testing, 147–152
CloudMonkey, 36, 149
Cognitive impairment, accessibility testing, 61–62
Cohn, Mike, 105, 106
Color deficiencies. *See* Visual impairment.
Color-blindness. *See* Visual impairment.
Communication, tester skill, 179–181
Communities, 196
Community support, 174–175
Conferences, 195–196
Configuration data, mobile test strategy, 165
Connected cars, 201–203
Connected homes, 200–201
Constant learning, tester skill, 183–184
Cooper, Alan, 30
Coordinate-based recognition tools, 110
COP FLUNG GUN (Communication, Orientation, Platform, Function, Location, User scenarios, Notifications, Guidelines, Gesture, Updates, Network), 92
Crash reports, 176
Crashes on specific devices, tester skills improvement, 187–192
Creativity, tester skill, 184

Critical thinking, tester skill, 181–182
Crowd testing, 141–147
Crowd testing providers, 145
crowdsourcedtesting, 145
Crowdsourcing, 141
CruiseControl, 137
CSS3, 22
Cucumber for Android, 125–126
Curiosity, tester skill, 181
Customer focus, tester skill, 184–185
Customers. *See also* Users.
  getting information about, 29–32
  inviting to usability testing sessions, 31–32
  representing with personas, 30–32
  satisfying, checklist for, 32

Databases, testing, 75
Date formats, testing, 46
Deafness. *See* Auditory impairment.
Decision tables, testing, 55
Deleting apps, reasons for, 2
Description, in bug reports, 97
Developing apps. *See* App development.
Device movement, detecting, 38
Device orientation, detecting, 38
Dock section, Android and iOS devices, 18
Dolphin Web browser *vs.* other browsers, 48
Double tap gesture, 42
Downloading resources. *See* Online resources.
Drag gesture, 42
Dumb phones, 5
Dutch Testing Days, 195
Dynamic testing, 53–55

EDGE (Enhanced Data rates for Global Evolution), 13
eggPlant
  for Android, 129
  image recognition, 110
  OCR/text recognition, 111
Emulators
  Android Emulator, 51
  handling fragmentation, 35
  mobile device test clouds, 35
  pros and cons, 115–116
  vs. real devices, 51–52
  vs. simulators, 51–52
Energy Usage Instrument, 65–66
Espresso
  for Android, 128
  native object recognition, 111
ETSI (European Telecommunications Standards Institute) standards, 82
EuroSTAR, 195
Expected result, in bug reports, 98
Experitest
  for Android, 129
  for iOS, 134
  mobile cloud testing, 149
Explore It!, 196
ExtSolo, 122–123

FCC CUTS VIDS (Feature tour, Complexity tour, Claims tour, Configuration tour, User tour, Testability tour, Scenario tour, Variability tour, Interoperability tour, Data tour, Structure tour), 89–90, 170
Feature phones, 5
Fiddler battery tool, 64
FindBugs tool, 54
Firefox Mobile browser *vs.* other browsers, 48
Fitness wristbands, 203–204

5G mobile standard, 13
4G networks, 13
Fragmentation, definition, 5
Fragmentation, handling
    emulators, 35
    grouping test devices, 33–34
    mobile cloud testing, 35–36
    mobile device labs, 34–36
    prioritizing mobile device groups, 33–34
    privacy, 35
    renting test devices, 34–36
    security, 35
    simulators, 35
Frank, 134
Freemium business model, 23–24
Fully charged battery, testing, 63–64
Functional testing, 56, 162

Gathering customer information, 30
Gecko, 47
General Packet Radio Service (GPRS), 13
Generations of mobile devices, 11–12
Geographical regions, adapting to. *See* L10n (localization).
Gesture support for Android, 125
Gestures
    detecting, 40
    testing, 41–42
Gherkin for Android, 126
GHUnit, 134
Global App Testing, 145
Google, Our Mobile Planet, 34
Google Best Practices, 57
Google Glass, 204–205
Google Instrumentation Test Runner, 128

Google Play native App Beta Testing, 138
Google Play Store, 25
Google Play store Guidelines, 26
Google Test Automation Conference, 195
Google wearables, 204
GPRS (General Packet Radio Service), 13
GPS coordinates, detecting, 39–40
GPS location, setting for Android, 122
Grouping test devices, 33–34
GSM (Global System for Mobile Communications), 12
Guidelines. *See also* Checklists; Mind maps; Mnemonics.
  Amazon Appstore Guidelines, 26
  Android, 81
  app development, 2–3
  app stores, review and publishing, 26
  Apple App Store Guidelines, 26
  Apple iOS approval, 81
  BlackBerry, 81
  BlackBerry World Guidelines, 26
  for corporate look-and-feel, 59
  Google Play store Guidelines, 26
  platform-specific mobile testing, 81–82
  test automation, 114–115
  testing connected cars, 202
  Windows Phone, 81
Guidelines, accessibility testing
  for Android, 62
  for iOS, 62
  WC3 initiative, 62
  for Windows Phone, 63
Gyroscope sensor, testing, 38

Hardware usage testing, 192
Heads, cell phones, 12

Health apps, 205–207
Hearing impairment. *See* Auditory impairment.
Help. *See* Books and publications; Online resources.
HockeyApp, 138
HockeyKit, 139
HomeKit, 201
How Google Tests Software, 196
Howe, Jee, 141
HTC Dream smartphone, 14
HTML5, 22
Humidity sensor, testing, 39
Hybrid apps, 20–21

I SLICED UP FUN (Inputs, Store, Location, Interactions/interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network), 90–92, 170
I18n (internationalization), 45–46
IEEE (Institute of Electrical and Electronics Engineers), 82
Ignite, 195
Image recognition tools, 109
iMobiTrax, 177
In-app purchases, 24
The Inmates Are Running the Asylum, 30
Inputs, Store, Location, Interactions/interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network (I SLICED UP FUN), 90–92, 170
Installation testing, 71–72
Instruments, energy usage function, 65
Internationalization (I18n), 45–46
Internet connections, testing, 4
Internet Explorer Mobile Web browser *vs.* other browsers, 48
Internet of Things (IoT). *See* IoT (Internet of Things).
Interrupt testing, 66–68
In-the-wild testing, 167–169

iOS
  accessibility testing, 62
  approval guidelines, 81
  design guide, 18
  design resources, 81
  generating pseudo-random test events, 67
  simulators, 52
iOS apps
  test automation. *See* Test automation, iOS.
  testing criteria, 103
iOS smartphone
  vs. Android, 17–19
  battery status, displaying, 18
  dock section, 18
  generating pseudo-random test events, 67
  Instruments energy usage function, 65
  interrupt testing, 67–68
  logging energy usage, online resource, 66
  network strength, displaying, 18
  notifications, displaying, 18
  pinning apps, 18
  status bar, 18
  stress testing, 67–68
  time, displaying, 18
  UI AutoMonkey tool, 67–68
  user interface, 18
ios-driver, 132–133
IoT (Internet of Things)
  Android Auto, 201
  Apple CarPlay, 201
  Apple HomeKit, 201
  connected car, 201–203
  connected home, 200–201

mobile testing, 199–203
Nest Labs, 200
overview, 199–200
ISTQB class, 55

Jamo Solutions
Android, 129
iOS, 134
Janky, 137
Java static analysis tools, 54
JavaScript, 22
Jenkins, 137
Johnson, Karen Nicole
mind maps, 93–96
mnemonics, 88
JouleUnit, energy profiling tool, 65
JSON Wire Protocol
compatibility with Selendroid, 124
and ios-driver, 132–133
JUnit tool, 54

Kaner, Cem, 183
Kelly, Mike, 89
Keyboard input, testing, 44
Keyboards, testing, 187–188
Keynote
for Android, 129
for iOS, 134
mobile cloud testing, 149
Keynote Mobile Testing, 36
KIF (Keep It Functional), 133
KIFSU (Keep It Fast Simple Usable) principle, 3
KISS (Keep It Simple, Stupid) principle, 3, 102
Kiwi, 134

Kohl, Jonathan, 90, 92

L10n (localization), 45–46
Language
  in bug reports, 100
  switching, for Android, 122
Languages
  adapting to. *See* I18n (internationalization); L10n (localization).
  fallback, 46
Launch strategy. *See* Mobile launch strategy.
Lessons Learned in Software Testing, 196
Let’s Test, 195
Lint tool, 54
Local storage testing, 76
Localization (L10n), 45–46
Location sensor, testing, 39–40
Log files
  in bug reports, 101–102
  checking, 82–83
Logging energy usage in iOS, 66
Login task, testing, 190
Long touch gesture, 41
Low battery, testing, 64
LTE (Long Term Evolution), 13

Magazines. *See* Books and publications.
Magnetic fields, detecting, 38–39
Magnetic sensor, testing, 38–39
Manual testing, mobile test strategy, 162, 167–169
Maximum device scope, 160
Metrics, mobile test strategy, 162
mHealth apps, testing, 205–207
Microphones, testing, 42–43
Microsoft Store, 26

Mind maps, 93, 94–96
Ministry of Testing, 93
Mnemonics. *See also* Checklists; Guidelines; Mind maps.
  COP FLUNG GUN (Communication, Orientation, Platform, Function, Location, User scenarios, Notifications, Guidelines, Gesture, Updates, Network), 92
  FCC CUTS VIDS (Feature tour, Complexity tour, Claims tour, Configuration tour, User tour, Testability tour, Scenario tour, Variability tour, Interoperability tour, Data tour, Structure tour), 89–90
  I SLICED UP FUN (Inputs, Store, Location, Interactions/interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network), 90–92
  mobile test strategy, 170
  overview, 87
  SFDPOT (Structure, Function, Data, Platform, Operations, Time), 88
Mob4Hire, 145
Mobile, definition, 11
Mobile App Europe, 195
Mobile Cheat Sheet, 79
Mobile cloud testing
  handling fragmentation, 35–36
  mobile device test clouds, 35–36
Mobile cloud testing providers, 149–150
Mobile community, tester skills improvement, 194–195
Mobile device, in bug reports, 99
Mobile device labs, 34–36
Mobile device test clouds
  emulators, 35
  grouping test devices, 33–34
  mobile cloud testing, 35–36
  mobile device labs, 34–36
  prioritizing mobile device groups, 33–34
  privacy, 35
  renting test devices, 34–36

security, 35
simulators, 35
Mobile devices. *See also* Smartphones; *specific brands*.
acquiring for testing, 6–8
cell phones, 4–5
dumb phones, 5
feature phones, 5
fragmentation, 5
renting for testing, 7
selecting for testing, 6
Mobile Frameworks Comparison Chart, 21
Mobile Labs, 36
Mobile launch strategy
check release material, 172–174
community support, 174–175
crash reports, 176
post-release, 174
pre-launch, 172–174
reviews, 175
statistics, 176–177
timing of the release, 173
tracking, 176–177
Mobile networks, history of
AMPS (Advanced Mobile Phone Service), 12
AMTS (Advanced Mobile Telephone Service), 12
CDMA (Code Division Multiple Access), 12
EDGE (Enhanced Data rates for Global Evolution), 13
generations, 11–12
GPRS (General Packet Radio Service), 13
GSM (Global System for Mobile Communications), 12
LTE (Long Term Evolution), 13
Mobile Radio Telephone System, 12
MTD (Mobile Telephony System D), 12

MTS (Mobile Telephone Service), 12
NMT (Nordic Mobile Telephone), 12
OLT (Offentlig Landmobil Telefoni), 12
overview, 11–14
speed, 11–13
UMTS (Universal Mobile Telecommunications System), 13
WiMAX (Worldwide Interoperability for Microwave Access), 13
Mobile Personas, 30
Mobile platform, in bug reports, 99
Mobile QA Zone, 196
Mobile Radio Telephone System, 12
Mobile release cycles, 6–8
Mobile Security Testing, 79
Mobile Telephone Service (MTS), 12
Mobile Telephony System D (MTD), 12
Mobile test strategy
automated testing, 162
black box testing, 164
configuration data, 165
defining requirements, 156–158
FCC CUTS VIDS, 170
functional testing, 162
I SLICED UP FUN, 170
in-the-wild testing, 167–169
manual testing, 162, 167–169
mnemonics, 170
mobile checklists, 169–170
mobile tours, 169–170
nonfunctional testing, 162–163
overview, 155–156
possible metrics, 162
product risks, 172
regression testing, 162

stable data, 165
target devices, selecting, 166–167
target group information, gathering, 157–158
temporary data, 165
test automation, 170–171
test data, 164–166
test environment, selecting, 166–167
test levels, 161–163
test techniques, 163–164
usage scenarios, 168
white box testing, 163
Mobile test strategy, scope of testing
maximum device scope, 160
multidevice scope, 159–160
single-device scope, 159
use-case scope, 160–161
Mobile testing. *See also* Quality assurance measures; *specific tests*.
acquiring devices for, 6–8
vs. automated testing, 52–53
batteries. *See* Battery usage, testing.
be creative, 83
boundary value testing, 55
cameras, 43–44
checking log files, 82–83
CI (continuous integration), 136–137
cloud testing, 147–152
connected car, 201–203
for connected cars, 202
connected home, 200–201
crowd testing, 141–147
databases, 75
decision tables, 55
documenting results. *See* Bug reports.

ETSI (European Telecommunications Standards Institute) standards, 82
functional testing, 56
generating pseudo-random test events for iOS smartphone, 67
generic checklist for, 85–87
gestures, 41–42
hardware usage, 192
IEEE (Institute of Electrical and Electronics Engineers), 82
installation testing, 71–72
interrupt testing, 66–68
IoT (Internet of Things), 199–203
keyboard input, 44
keyboards, 187–188
local storage testing, 76
login task, 190
mHealth apps, 205–207
microphones, 42–43
mind maps, 93, 94–96
on multiple platforms, 187–192
vs. other technologies, 1–2, 8
payment task, 190
for people with disabilities. *See* Accessibility testing.
performance. *See* Performance testing.
permissions, 190–191
platform-specific guidelines, 81–82
preinstalled apps, 44
pseudo-random test events for Android, 66
quality of Internet connections, 4
in real life environments, 3–4
remote locations, 4
renting devices for, 7
security. *See* Security testing.
selecting devices for, 6
smart glasses, 204–205

sound inputs, 42–43
standards conformance, 82
standby testing, 71
stress testing, 66–68
system apps, 44
Testing Map, 93
touchscreen, 41–42
uninstall testing, 72
user interface. *See* Usability testing.
voice commands, 42–43
W3C (World Wide Web Consortium), 82
wearables, 203–204
weather conditions, 4
widgets, 187–189

Mobile testing, mnemonics. *See also* Checklists; Guidelines; Mind maps.
COP FLUNG GUN (Communication, Orientation, Platform, Function, Location, User scenarios, Notifications, Guidelines, Gesture, Updates, Network), 92
FCC CUTS VIDS (Feature tour, Complexity tour, Claims tour, Configuration tour, User tour, Testability tour, Scenario tour, Variability tour, Interoperability tour, Data tour, Structure tour), 89–90
I SLICED UP FUN (Inputs, Store, Location, Interactions/interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network), 90–92
overview, 87
SFDPOT (Structure, Function, Data, Platform, Operations, Time), 88

Mobile testing, sensors and interfaces
acceleration sensor, 38
ambient light sensor, 37
changes in device movement, detecting, 38
device orientation, detecting, 38
gestures, detecting, 40
GPS coordinates, detecting, 39–40
gyroscope sensor, 38

humidity sensor, 39
location sensor, 39–40
magnetic fields, detecting, 38–39
magnetic sensor, 38–39
possible tests, 40–41
pressure sensor, 39
proximity sensor, 37–38
screen brightness, 37
temperature sensor, 39
touchless sensor, 40
Mobile testing, update testing
adb tool, 74–75
for Android apps, 74–75
for iOS apps, 73–74
overview, 72
typical scenarios, 73
Mobile Tools, 79
Mobile tours, 169–170
Mobile tracking tools, 177
Mobile Usability, 57
Mobile Web browsers
comparison of, 48
layout engines, 47
MobileAppTracking, 177
Mobile-device-specific information, in bug reports, 99
Mobility and data networks, 3–4
Monkey tool, 66
MonkeyTalk
for Android, 129
coordinate-based recognition, 110
for iOS, 134
OCR/text recognition, 111
Moolya, 92

MTD (Mobile Telephony System D), 12
MTS (Mobile Telephone Service), 12
Multidevice scope, 159–160
Multitouch gesture, 42

Native apps, description, 19–20
Native Object Recognition, 111
Neotys, 149
Nest Labs, 200
Network condition and environment, in bug reports, 100
Network strength, displaying, 18
99tests, 145
NMT (Nordic Mobile Telephone), 12
Nonfunctional testing, 162–163
Notifications, displaying, 18

OCMock, 134
OCR/text recognition, 110–111
OLT (Offentlig Landmobil Telefoni), 12
1G networks, 12
Online resources. *See also* Books and publications.
    5G mobile standard, 13
    99tests, 145
    adjust, 177
    Amazon Appstore Guidelines, 26
    Android design guide, 18
    Android Emulator, 51
    Android guidelines, 81
    App Quality Alliance site, 103
    app rejection, reasons for, 18
    app store review and publishing guidelines, 26
    Appaloosa, 138
    AppBlade, 138
    Appium, 134

Applause, 138, 145
Apple App Store, 25
Apple App Store Guidelines, 26
Apple iOS approval guidelines, 81
Apple iOS design resources, 81
Apple Watch, 204
“Applying the SFDPOT Heuristic to Mobile Testing,” 88
appsfire, 177
AppsFlyer, 177
AppThwack, 36, 149
Appurify, 149
Bamboo, 137
battery usage, 65–66
Beta by Crashlytics, 138
beta distribution tools, 138–139
BirdFlight, 138
BlackBerry guidelines, 81
BlackBerry World, 26
BlackBerry World Guidelines, 26
Blink, 47
Books about SQLite, 75
Bots, 137
Buildbot, 137
CDMA (Code Division Multiple Access), 12
CI systems, 137
Clicktale, 177
CloudMonkey, 36, 149
crowd testing providers, 145
crowdsourcedtesting, 145
CruiseControl, 137
CSS3, 22
EDGE (Enhanced Data rates for Global Evolution), 13
eggPlant, 129

Energy Usage Instrument, 65–66
Espresso, 128
Experitest, 129, 134, 149
Frank, 134
gathering customer information, 30
Gecko, 47
GHUnit, 134
Global App Testing, 145
Google Best Practices, 57
Google Instrumentation Test Runner, 128
Google Play native App Beta Testing, 138
Google Play Store, 25
Google Play store Guidelines, 26
Google wearables, 204
GPRS (General Packet Radio Service), 13
GSM (Global System for Mobile Communications), 12
HockeyApp, 138
HockeyKit, 139
HTML5, 22
iMobiTrax, 177
iOS design guide, 18
iOS simulators, 52
ios-driver, 133
Jamo Solutions, 129, 134
Janky, 137
JavaScript, 22
Jenkins, 137
Keynote, 129, 134, 149
Keynote Mobile Testing, 36
Kiwi, 134
logging energy usage in iOS, 66
Microsoft Store, 26
Mob4Hire, 145

mobile cloud testing providers, 149–150
mobile device test cloud, 36
Mobile Frameworks Comparison Chart, 21
Mobile Labs, 36
Mobile Personas, 30
mobile tracking tools, 177
mobile usability, 57
mobile usability design principles, 57
mobile user interface design principles, 57
MobileAppTracking, 177
MonkeyTalk, 129, 134
Neotys, 149
OCMock, 134
Open Device Lab, 36
operating system market shares, 16
Our Mobile Planet, 34
passbrains, 145
Pebble, 204
Perfecto Mobile, 36, 129, 134, 150
PhoneGap framework, 20
platform-specific guidelines, 81–82
Presto, 47
Ranorex, 129, 134, 150
Roboelectric, 129
Robotium project pages, 122
Robotium wiki page, 122
Sauce Labs, 36, 150
Siesta, 129
Silk Mobile, 129, 134
SOASTA, 129, 134
Specta, 134
Subliminal, 134
TeamCity, 137

TenKod EZ TestApp, 129
TestBirds, 145
TestChameleon, 150
testCloud, 145
Testdroid, 150
TestFlight, 139
testing criteria for Android and iOS, 103
testing tools, 54
Testmunk, 36, 150
TestObject, 36, 129, 150
TestPlus, 145
TouchTest, 36
Travis CI, 137
Trident, 47
UI Automator, 129
UMTS (Universal Mobile Telecommunications System), 13
usability heuristics, 57
usability principles and techniques, 57
usability testing, 57
wearables, 204
Web browser layout engines, 47
WebKit, 47
widgets, 18
WiMAX (Worldwide Interoperability for Microwave Access), 13
Windows Phone guidelines, 81
Xamarin Test Cloud, 36, 150
XCTest, 134
Zucchini, 134
Online resources, for testers
Agile Testing Days, 195
Association for Software Testing, 196
Belgium Testing Days, 195
blogs, 196–197

books and publications, 196–197
communities, 196
conferences, 195–196
Dutch Testing Days, 195
EuroSTAR, 195
Google Test Automation Conference, 195
Ignite, 195
Let's Test, 195
Mobile App Europe, 195
Mobile QA Zone, 196
Øredev, 195
Software Testing Club, 196
STAREAST, 196
STARWEST, 196
TestBash, 196
TestExpo, 196
Testing Circus, 196
uTest community, 196
Open Device Lab, 36
Opera Mobile Web browser *vs.* other browsers, 48
Operating systems
the big two, 16–19. *See also* Android; iOS.
in bug reports, 99
relative market shares, 16
Øredev, 195
Our Mobile Planet, 34
Overheating, batteries, 64
OWASP (Open Web Application Security Project), 77
OWASP Mobile Security Page
Mobile Cheat Sheet, 79
Mobile Security Testing, 79
Mobile Tools, 79
Top 10 Mobile Risks, 80

Paid business model, 24
passbrains, 145
Payment task, testing, 190
Pebble, 204
Perfecto Mobile
  for Android, 129
  coordinate-based recognition, 110
  for iOS, 134
  mobile cloud testing providers, 150
  mobile device test cloud, 36
Performance testing
  critical areas, 69
  mobile apps, 70–71
  overview, 68–69
  reproducing issues, 189–190
Permissions, testing, 190–191
Personas, 30–32
PhoneGap framework, 20
Physical impairment, accessibility testing, 61
Pinch open/close gestures, 42
Pinning apps, 18
Platform-specific guidelines for testing, 81–82
PMD tool, 54
Post-release, mobile launch strategy, 174
Preinstalled apps, testing, 44
Pre-launch, mobile launch strategy, 172–174
Pressure sensor, testing, 39
Presto, 47
Prioritizing mobile device groups, 33–34
Privacy, 35
Process focus, 53–55
Product focus, 53–55
Product risks, mobile test strategy, 172

Professional Tester, 197
Programming language, choosing, for iOS tests, 132–133
Programming skills, tester skill, 185
Proximity sensor, testing, 37–38
Pyramid. *See* Test automation pyramid.

Quality assurance measures. *See also* Mobile testing.
  black box testing, 54–55
  dynamic testing, 53–55
  finding bugs, 53–55
  Java static analysis tools, 54
  process focus, 53–55
  process focus phase, 54
  product focus, 53–55
  protecting from bugs, 53–55
  static testing, 53–55
  white box testing, 54

Ranorex
  for Android, 129
  for iOS, 134
  mobile cloud testing providers, 150
Real devices, pros and cons of testing, 116–117
Reda, Renas, 122
Regression testing, mobile test strategy, 162
Rejecting apps, reasons for, 18
Release cycles. *See* Mobile release cycles.
Release material, checking, 172–174
Renting devices for testing, 7
Renting test devices, 34–36
Reproducible, in bug reports, 99
Requirements definition, mobile test strategy, 156–158
Research in Motion, 14
Resistive touchscreens, 41

Review and publishing guidelines, app stores, 26
Reviews, mobile launch strategy, 175
Roboelectric, 129
Robotium
  for Android, 121–122
  native object recognition, 111
  OCR/text recognition, 111
  recording test scripts, 122–123
Robotium Extensions, 122–123
Robotium project pages, 122
Robotium Recorder, 122–123
Robotium wiki page, 122
Rotate gesture, 42
Rotating touchscreens, 42
RoutineBot, 110

Safari Web browser vs. other browsers, 48
Samsung app store, 25
Sauce Labs, 36, 150
Scope of testing
  maximum device scope, 160
  multidevice scope, 159–160
  single-device scope, 159
  use-case scope, 160–161
Screen brightness, testing, 37
Screens. *See* Touchscreens.
Screenshot or video, in bug reports, 101
Security
  handling fragmentation, 35
  mobile device test clouds, 35
Security testing
  cache storage, 78
  client-side validation, 78
  common app security mistakes, 77–79

encryption, 78
fuzzing, 77
overview, 76–77
penetration testing, 77
security requirements analysis, 79–80
summary, 80–81
testing tools, 77
Top 10 Mobile Risks, 80
unencrypted communications, 78
unintended local storage, 78
unnecessary app permissions, 78
SeeTest, 111
Selendroid
for Android, 124–125
native object recognition, 111
Selenium, 111
Selling advertisements, 24
Sensors, testing. *See* Mobile testing, sensors and interfaces.
Severity of error, in bug reports, 101
SFDPOT (Structure, Function, Data, Platform, Operations, Time), 88
Sherry, Rosie, 93–96
Siesta, 129
Sikuli, 110
Silk Mobile
for Android, 129
for iOS, 134
Simulators
vs. emulators, 51–52
handling fragmentation, 35
iOS, 52
mobile device test clouds, 35
pros and cons, 115–116
vs. real devices, 51–52

Single-device scope, 159
Smart glasses, 204–205
Smart watches, 203–204
Smartphones. *See also* Mobile devices; *specific brands*.
    common features, 5
    external components, 15–16
    hardware components, 15
    history of, 14–16
    market penetration, 5
    post-2007, 14–16
SOASTA
    for Android, 129
    for iOS, 134
Software build version, in bug reports, 100
Software Testing Club, 196
Sound inputs, testing, 42–43
Specification by Example, 196
Specta, 134
Speed of mobile networks, history of, 11–13
Spoon, 123–124
Square, 123, 133
Stable data, mobile test strategy, 165
Standards conformance, testing, 82
Standby testing, 71
STAREAST, 196
STARWEST, 196
Static analysis tools, 54
Static testing, 53–55
Statistics, mobile launch strategy, 176–177
Status bar, 18
Steps to reproduce, in bug reports, 98
Strategy
    for product launch. *See* Mobile launch strategy.

for testing. *See* Mobile test strategy.
Stress testing, 66–68
Structure, Function, Data, Platform, Operations, Time (SFDPOT), 88
Subliminal, 134
Swipe gesture, 41
System apps, testing, 44

Tap gesture, 41
Tap into Mobile Application Testing, 92, 196
Target device selection, mobile test strategy, 166–167
Target group information gathering, mobile test strategy, 157–158
TeamCity, 137
Tea-time with Testers, 197
Technical skills, tester skill, 185
Temperature sensor, testing, 39
Temporary data, mobile test strategy, 165
Tenacity, tester skill, 183
TenKod EZ TestApp
for Android, 129
native object recognition, 111
Test automation
cloud-based, 115–116, 152
emulators, pros and cons, 115–116
goals for, 113–114
guidelines for, 114–115
vs. manual testing, 52–53
mobile test strategy, 162, 170–171
vs. mobile testing, 52–53
real devices, pros and cons, 116–117
selecting tasks for, 113–115
simulators, pros and cons, 115–116
Test automation pyramid
default configuration, 105–106
flipped configuration, 106–108

mobile test configuration, 108–109
Test automation tools
Appium, 111
beta distribution, 138–139
Calabash, 111
capture and replay, 111–112
coordinate-based recognition, 110
eggPlant, 110, 111
Espresso, 111
image recognition, 109
MonkeyTalk, 110, 111
Native Object Recognition, 111
OCR/text recognition, 110–111
overview, 109
Perfecto Mobile, 110
recommendations, 112–113
Robotium, 111
RoutineBot, 110
SeeTest, 111
selection criteria, 118–120
Selendroid, 111
Selenium, 111
Sikuli, 110
TenKod EZ TestApp, 111
TestObject, 110, 111
Test automation tools, Android
Advanced User Interactions API, 125
Appium, 127–128
automating UI acceptance tests, 125
Calabash for Android, 125–126
Cucumber, 125–126
eggPlant, 129
Espresso, 128

Experitest, 129
ExtSolo, 122–123
gesture support, 125
Gherkin, 126
Google Instrumentation Test Runner, 128
Jamo Solutions, 129
Keynote, 129
MonkeyTalk, 129
overview, 120–121
Perfecto Mobile, 129
Ranorex, 129
recommendations, 129–130
recording test scripts, 122–123
Roboelectric, 129
Robotium, 121–122
Robotium Extensions, 122–123
Robotium Recorder, 122–123
Selendroid, 124–125
setting GPS location, 122
Siesta, 129
Silk Mobile, 129
SOASTA, 129
Spoon, 123–124
switching current language, 122
TenKod EZ TestApp, 129
TestObject, 129
toggling Wi-Fi on and off, 122
UI Automator, 129
UI Automator Viewer, 121
Test automation tools, iOS
Appium, 134
Calabash for iOS, 131–132
choosing a programming language, 132–133

Experitest, 134
Frank, 134
GHUnit, 134
ios-driver, 132–133
Jamo Solutions, 134
Keynote, 134
KIF (Keep It Functional), 133
Kiwi, 134
MonkeyTalk, 134
OCMock, 134
Perfecto Mobile, 134
Ranorex, 134
Silk Mobile, 134
SOASTA, 134
Specta, 134
Subliminal, 134
UI Automation, 129–130
XCTest, 134
Zucchini, 134
Test data
in bug reports, 100
mobile test strategy, 164–166
Test environment selection, mobile test strategy, 166–167
Test levels, mobile test strategy, 161–163
Test scripts, recording, 122–123
Test strategy. *See* Mobile test strategy.
Test techniques, mobile test strategy, 163–164
TestBash, 196
TestBirds, 145
TestChameleon, 150
testCloud, 145
Testdroid, 150
Testers

blogs, 196–197
books and publications, 196–197
communities, 196
conferences, 195–196
named in bug reports, 102
success factors, 207–209
Testers, required skills
communication, 179–181
constant learning, 183–184
creativity, 184
critical thinking, 181–182
curiosity, 181
customer focus, 184–185
programming skills, 185
technical skills, 185
tenacity, 183
Testers, skills improvement
crashes on specific devices, 187–192
engage with the mobile community, 194–195
learn from other apps, 186
observe other people, 192–193
participate in testing competitions, 193–194
Twitter, 194
TestExpo, 196
TestFlight, 139
Testing apps. *See* Mobile testing.
Testing Circus (community), 196
Testing Circus (magazine), 197
Testing competitions, tester skills improvement, 193–194
Testing criteria for Android and iOS, 103
Testing Map, 93
Testing tools
adjust, 177

appsfire, 177
AppsFlyer, 177
for automation. *See* Test automation tools.
Checkstyle, 54
Clicktale, 177
FindBugs, 54
iMobiTrax, 177
JUnit, 54
Lint, 54
for mobile tracking, 177
MobileAppTracking, 177
PMD, 54
static analysis, 54
TestNG, 54
white box testing, 54
XCTest, 54
Testmunk, 36, 150
TestNG tool, 54
TestObject
for Android, 129
image recognition, 110
mobile cloud testing providers, 150
mobile device test cloud, 36
OCR/text recognition, 111
TestPlus, 145
3G networks, 13
Time, displaying, 18
Timing of the release, 173
T-Mobile G1 smartphone, 14
Tools. *See* Online resources; Testing tools; *specific tools*.
Top 10 Mobile Risks, 80
Touch gestures, 41–42
Touchless sensor, testing, 40

Touchscreens
- capacitive, 41
- effects of weather conditions, 42
- mobile testing, 41–42
- resistive, 41
- rotating, 42
- testing, 41–42
- touch gestures, 41–42
- types of, 41

TouchTest, 36
Tracking, mobile launch strategy, 176–177
Tracking tools, 177
Transaction business model, 24
Transceivers, cell phones, 12
Travis CI, 137
Trident, 47
Twitter, tester skills improvement, 194
2G networks, 12–13

UI acceptance tests, automating, 125
UI Automation, 129–130
UI Automator, 129
UI Automator Viewer, 121
UMTS (Universal Mobile Telecommunications System), 13
Uninstall testing, 72
Update testing
- adb tool, 74–75
- for Android apps, 74–75
- for iOS apps, 73–74
- overview, 72
- typical scenarios, 73

Uploading apps to app stores, 26
Usability
- heuristics, 57

The Inmates Are Running the Asylum, 30
principles and techniques, 57
Usability testing
allow mistakes, 58
consistency, 59
corporate identity guidelines for look-and-feel, 59
details, checking, 58
document your observations, 59–60
inviting customers to, 31–32
less is more, 57
navigation, checking, 58
overview, 56
self-explanatory apps, 57
usability and design patterns, 58
verify text, 59
workflows, checking, 58
Usability testing sessions, 31–32
Usage scenarios, mobile test strategy, 168
Usage statistics, battery life, 64–66
Use-case scope, 160–161
User interface
Android *vs.* iOS, 18
components, viewing properties of, 121
design principles for mobile devices, 57
testing. *See* Mobile testing, sensors and interfaces.
Users, expectations. *See also* Customers.
overview, 2–3
priority app characteristics, 2
reasons for deleting apps, 2
uTest community, 196

Visual impairment, accessibility testing, 60–61
Voice commands, testing, 42–43

W3C (World Wide Web Consortium)
  accessibility guidelines initiative, 62
  conformance testing, 82
Wearables
  fitness wristbands, 203–204
  Google Glass, 204–205
  mobile testing, 203–204
  overview, 203
  smart glasses, 204–205
  smart watches, 203–204
Weather conditions
  effects on touchscreens, 42
  testing, 4
Web apps, description, 22–23
Web browser layout engines, 47
Web browsers. *See* Mobile Web browsers.
WebKit, 47
Weekend Testing, 193–194
White box testing
  mobile test strategy, 163
  tools for, 54
Widgets
  iOS *vs.* Android, 18
  testing, 187–189
Wi-Fi, toggling on and off, 122
WiMAX (Worldwide Interoperability for Microwave Access), 13
Windows Phone
  accessibility guidelines, 63
  accessibility testing, 63
  app store, 26
  relative market share, 16
Windows Phone apps, downloading, 26
Work-around, in bug reports, 98

Worldwide Interoperability for Microwave Access (WiMAX), 13

Xamarin Test Cloud, 36, 150
XCTest, 134
XCTest tool, 54

0G networks, 12
Zucchini, 134

Addison Wesley
REGISTER
THIS PRODUCT

Code Snippets

```
/../daniel/android/sdk/build-tools/android-4.4/
```

```
./aapt d badging /daniel/myApp/myApp.apk | grep 'pack'
...
package: name='com.myApp' versionCode='' versionName=''
...
```

```
./adb shell monkey -p com.myApp -v 2000
```

```
config: {
  numberOfEvents: 2000,
  delayBetweenEvents: 0.05,    // In seconds

  // Events that will be triggered on the phone
  eventWeights: {
    tap: 30,
    drag: 1,
    flick: 1,
    orientation: 1,
    clickVolumeUp: 1,
    clickVolumeDown: 1,
    lock: 1,
    pinchClose: 10,
    pinchOpen: 10,
    shake: 1
  },

// Probability that touch events will have these different properties
  touchProbability: {
    multipleTaps: 0.05,
    multipleTouches: 0.05,
    longPress: 0.05
  }
},
./adb install RELEASED_APP_NAME.apk
```

```
./adb install -r NEW_VERSION_APP.apk
```

```
/android/sdk/tools/uiautomatorviewer.sh
```

```
/* Robotium will click on the text "Welcome" */
solo.clickOnText("Welcome");
/* Robotium will enter the string MySecretPassword into the input
field with the ID 2 */
solo.enterText(2, "MySecretPassword");
/* Robotium will click on the button with the label "Login" */
solo.clickOnButton("Login");
/* Robotium will simulate a click on the native back button */
solo.goBack();
```

```
Spoon.screenshot(activity, "Login_Screen");
assertThat(password).hasNoError();
instrumentation.runOnMainSync(new Runnable() {
       @Override public void run() {
                      password.setText("MySecretPassword");
       }
});
```

```java
WebElement loginButton = driver().findElement(By.id("startLogin"));
WebElement passwordInput = driver().findElement(By.id("password"));
passwordInput.sendKeys("MySecretPassword");
loginButton.click();
```

```
Feature: As a user I want to login
  Scenario: Login using valid credentials
       Given I am on the login screen
       When I enter "Username" into the user field
       And I enter "PWD" into the password field
       And I click the login button
       Then I must see my user account
```

```
When(/^I enter "(.*?)" into the user field$/)
do | username |
       fill_in("IDUserName", :with => "username")
end
```

```
WebElement loginText = driver.findElement(By.name("TextLogin"));
assertEquals("TextLogin", loginText.getText());
WebElement loginTextView =
  driver.findElementByClassName("android.widget.TextView");
assertEquals("TextLogin", loginTextView.getText());
WebElement button =
  driver.findElement(By.name("Login"));
button.click();
```

```
onView(withId(R.id.login)).perform(click());
onView(withId(R.id.logout)).check(doesNotExist());
onView(withId(R.id.input)).perform(typeText("Hello"));
```

```
app.keyboard().typeString("Some text");
rootTable.cells()["List Entry 7"].tap();
alert.buttons()["Continue"].tap();
```

```
By button = By.id("Login");
WebElement loginButton = driver.findElement(button);
Assert.assertEquals(loginButton.getAttribute("name"), "Login");
loginButton.click();
```

```
[tester enterText:@"user one" intoViewWithAccessibilityLabel: @"User
Name"];
[tester enterText:@"Mypassword" intoViewWithAccessibilityLabel:
@"Login  Password"];
[tester tapViewWithAccessibilityLabel:@"Login"];
```