BY Julian Harty & Marc van't Veer

Testing

Introduction

In practice, testing software has often been performed as a distinct one-time activity, after the code has been written and before the code has been released. Over the last decade testing industry leaders have seen the advantages of starting testing earlier in the project lifecycle, for instance involving testing in the design phase. This is sometimes known as 'shift-left' testing. And more recently testing is being extended to incorporate information and feedback gleaned post-deployment when the software is live. This is becoming known as 'shift-right' testing. These concepts are a great fit for mobile apps where users provide feedback in reviews and where mobile analytics and crash analytics provide additional information about usage of the app.

So testing is becoming more of a mindset and a continual practice than a one-shot activity. Teams who use 'Agile' development are sometimes prone to limit testing of a feature to within a sprint which may constrain and box-in the testing. New testing approaches may need to emerge that take a more holistic approach, not limited to testing within a sprint. More needs testing than the new stories and features, and testing should not be limited to testing the code we write - testing of libraries, tools, services, approaches, etc. can all help improve the quality of our work and enable teams to make better, more informed decisions rather than relying on advertising, gut-feel, or cost.

"**But test them all; hold on to what is good**"[1]

Testing also needs to extend beyond the mobile app - our users expect no less! Apps often rely on third-party and cloud-based architectures and testing needs to factor these in. Users expect to be able to work seamlessly across multiple devices, for instance by creating an email with a photo attachment on their smartphone and then editing the email on their laptop using a web browser before sending it from yet another device and flavor of software. Therefore we need to consider how users may use our apps and whether the app extends beyond the mobile device. In parallel, mobile apps may be part of a larger ecosystem where smartwatches, fitness monitors, etc. are integrated and provide data to the app. Our testing needs to reflect the likely usage patterns by end users.

Finally for now, our apps need to cope well in an ever changing and often expanding microcosm. New releases of the platform, new devices, and new usage patterns can all expose weaknesses and limitations in an app. Unless we actively and continually pay attention our app will be left to fend for itself and may let down or disappoint our users.

Testing might be seen as an impediment but failures in your app can be all too public. And recovering your credibility is hard when your app has a poor score in the app store. Rather than waiting for users to decide the fate, testing your mobile apps can adjust the balance in your favor. By reading this chapter you have the opportunity to help equip you and your testing team so they can test your app more effectively.

This chapter covers the general topics; testing for specific platforms is covered in the relevant chapter.

1 1 Thessalonians 5:21

Beware of Differences

Platforms, networks, device settings, device models, and even firmware, are all specific and differ from each other. Any could cause problems for your applications[2]. This means we need a variety of devices to test on.

The range and variety of devices continues to accelerate, and users are increasingly connecting mobile devices together, for example IoT devices, wearables, cars or home appliances. More and more devices are needed for a testlab to create sufficiently representative test environments. Obtaining additional devices is important. From a user's perspective a mobile app is the combination of software, hardware and environment. They want and expect an app to work regardless of all these details. Furthermore, the boundaries of what needs to work extends even further, as many apps now interact with external devices and sensors. As an example, Disney have designed seamless experiences using smart wristbands[3] which integrate into a larger ecosystem. Devices can also be used to pay for travel[4] and shopping, cameras are used by banking software to identify and authorize cash withdrawals, and so on. Any problems, incompatibilities, and so on can adversely affect the UX and may have significant impact for instance if users are denied access to transport, money, and more.

A basic strategy for testing mobile apps is to assume that every combination is unique and different from another and will behave slightly differently. It would be impractical to test each combination, a more fruitful approach is to invest time

2 An example of device specific problems with Android on Samsung is anasambri.com/android/special-place-for-samsung-in-android-hell.html

3 wired.com/2015/03/disney-magicband

4 vodafone.nl/shop/mobiel/abonnement/extra-opties/smartlife/wallet/reizen

in learning what the impact of the differences are and testing a subset of the combinations that maximizes the insights and confidence we have in the behavior of the app across most of the combinations. Key skills include:

— Device analysis: What are the main differences? And when are these differences relevant to the app? (And which ones can we ignore).
— Extrapolation: What does a test on one device say to the thousand of devices out there in the wild?

Discovering Differences

There are several ways to identify the effects of specifics, for instance, a tester may notice differences in the performance of the app and the behavior of the UI when testing on different devices. Automation may also detect differences and can help you selecting devices that support the required features[5].

Conversely, Mobile Analytics can help identify differences in various aspects including performance and power consumption when the app is being used by many users on the vast variety of their devices. Some compelling examples of differences in behavior and on ways issues were addressed in a paper published by computer scientists from the University of Wisconsin[6]. A book is also available from HP Enterprise on the

5 mobiletestingblog.com/2017/05/30/optimizing-android-test-automation-development

6 "Capturing Mobile Experience in the Wild: A Tale of Two Apps", available as a download at static.googleusercontent.com/media/research.google.com/en//pubs/archive/41590.pdf

confluence between mobile analytics and testing for mobile apps[7].

Testing Needs Time - You Need a Strategy

The strategy defines how much test time is spent to the different parts of the mobile app and during the different development phases. There are tradeoffs on how best to spend whatever time you have. For instance, testing features in more detail versus testing on a wider variety of devices versus testing various quality aspects including performance, usability and security.

The conditions a mobile app need to operate are vast and to factor in these conditions into the testing is challenging. There may be more productive ways to obtain some information, for instance by using feedback from end-users and from mobile analytics. However, the risks of deferring data gathering (versus testing internally) need to be actively considered. An effective test strategy aims to balance both approaches. With the risk analysis, the quality perspectives and the available time the test plan can be created.

Continuous Testing

Continuous delivery needs continuous testing. Viable apps need to be updated on an ongoing basis. Updates may include fixes for new platform versions or device models, new functionality and other improvements. Therefore, testing is not a one-off task; high quality apps befit ongoing, optimized testing, including testing in production. Production testing

7 themobileanalyticsplaybook.com (co-written by Julian Harty, one of the authors of this chapter)

includes testing engagement and validation as well as early detection of potential problems before they mushroom.

Manage your Testing Time

Testing as you have discovered can take many hours, far more than you may want to do, particularly if you are close to a deadline such as a release date. There are various ways you can manage time spent in testing, in parallel testing can be made more interesting, rewarding, and more productive.

- **Reduce setup time**: Find ways to deploy apps quickly and efficiently. Implement mechanisms to provide the appropriate test data and configuration on both the mobile device and the relevant servers. Aim to have devices and systems 'ready to test'.
- **Reduce time needed for reporting & bug analysis**: Data, screenshots, and even video, can help make bugs easier and faster to investigate. Data can include logs, system configurations, network traffic, and runtime information. Commercial tools can record actions and screenshots to reduce the time and effort needed to report and reproduce problems.
- **Risk analysis**: You can use the risk analysis to decide how and when to allocate testing effort. Risks are hard to determine accurately by the tester or developer alone; a joint effort from all the stakeholders of the mobile app can help to improve the risk analysis. Sometimes, the mobile app tester is the facilitator in getting the product risk analysis in place.
- **Scaling testing**: Increasing the throughput of testing by scaling it, for instance using test automation, cloud-based test systems, and more humans involved in the testing can help increase the volume, and potentially the quality,

of the testing. Using static analysis tools to review code and other artifacts can also help the team to find and fix problems before the app is released.

Involve End-Users in your Testing

Development teams need a mirror to develop a useful mobile app. Early user feedback can provide that mirror. You do not need many end-users to have good feedback[8]. Bigger value is gained with early involvement, multiple users, regular sessions, and multiple smaller tests. Testers can guide and facilitate the end-user testing, for instance, by preparing the tests, processing log files and analyzing results. They can also retest fixes to the app.

Whenever others are involved in testing an app, they need ways to access and use the app. Web apps can be hosted online, perhaps protected using: passwords, hard-to-guess URLs, and other techniques. Installable apps need at least one way to be installed, for instance using a corporate app store or specialist deployment services.

When the app is closer to being production-ready, users can test the more mature version of the mobile app in Alpha & Beta tests phases. A development team or organization can offer an online community to give end-users early access to new releases, give loyalty points, ratings. This community should be a friendly ecosystem to receive feedback before the mobile app is released into the app store.

8 nngroup.com/articles/why-you-only-need-to-test-with-5-users

Proxy users

There are various services available that facilitate testing by other people. These people might be users of your app, however they are more likely to be proxy users, people who take the role of end users and provide additional perspective and perhaps insights into the behavior of an app. Possible sources of these testers include crowdsourcing[9]. Lookback[10] provides a different more personal flavor.

Effective Testing Practices

Testing, like other competencies, can be improved by applying various techniques and practices. Some of these need to be applied when developing your mobile app, such as testability, others apply when creating your tests, and others still when you perform your testing. Testdroid offers a good checklist[11] on getting the right testing expertise into your team.

Implementing Testability

Start designing and implementing ways to test your app while it is being developed. This applies especially for automated testing. For example, using techniques such as Dependency Injection in your code enables you to replace real servers (slow and flaky) with mock servers (controllable and fast). Use unique, clear identifiers for key UI elements. If you keep identifiers unchanged your automated tests require less maintenance.

9 Crowdtesting service providers include Applause.com, PassBrains.com, and TestBirds.de

10 lookback.io/

11 testdroid.com/testdroid/6336/get-the-superb-expertise-in-your-testingqa-team

Separate your code into testable modules. Several years ago, when mobile devices and software tools were very limited, developers chose to 'optimize' their mobile code into monolithic blobs of code, however the current devices and mobile platforms mean this form of 'optimization' is unnecessary and possibly counter-productive[12].

Provide ways to query the state of the application, possibly through a custom debug interface. You, or your testers, might otherwise spend lots of time trying to fathom out what the problems are when the application does not work as hoped.

Designing Test Environments

Test environments describe **where** we will perform the testing. It includes many facets. ISTQB defines test environment as: "An environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test."[13].

A key skill is to understand and be able to create suitable test environments. Often these include additional software tools and utilities, for instance to be able to read and filter log files, to control the network behavior, and/or to represent systems the app depends on including third-party authentication, payment processing, and so on. Like craftspeople of old, we are responsible for choosing the tools we use. Sometimes we may choose to create our own tools, for instance to mock the behaviors of an application server, to be able to inject errors, and so on.

12 To learn more about the reasons, read Google's blog: googletesting.blogspot.co.uk/2015/03/android-ui-automated-testing.html

13 istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html

Mnemonics and Tours

Mnemonics and Tours help us focus on **what** and **how** while we are testing a mobile app. Both concepts are heuristics, often useful, but also fallible.

A couple of mnemonics to help test mobile apps are:

— **I SLICED UP FUN**[14]: **I**nput (Test the application changing its orientation (horizontal/vertical) and trying out all the inputs including keyboard, gestures etc.), **S**tore (Use appstore guidelines as a source for testing ideas), **L**ocation (Test on the move and check for localization issues), **I**nteraction/Interruption (See how your app interacts with other programs, particularly built-in, native apps), **C**ommunication (Observe your app's behavior when receiving calls, e-mails, etc.), **E**rgonomics (Search for problem areas in interaction, e.g. small fonts), **D**ata (Test handling of special characters, different languages, external media feeds, large files of different formats, notifications), **U**sability (Look for any user actions that are awkward, confusing, or slow), **P**latform (Test on different OS versions), **F**unction (Verify that all features are implemented and that they work the way they are supposed to), **U**ser Scenarios (Create testing scenarios for concrete types of users), **N**etwork (Test under different and changing network conditions)

— **COP FLUNG GUN**[15] summarizes similar aspects under **C**ommunication, **O**rientation, **P**latform, **F**unction, **L**ocation, **U**ser Scenarios, **N**etwork, **G**estures, **G**uidelines, **U**pdates, **N**otifications.

14 kohl.ca/articles/ISLICEDUPFUN.pdf

15 moolya.com/blogs/2014/05/34/COP-FLUNG-GUN-MODEL

Karen Johnson provides helpful material on using heuristics and mnemonics for testing software at *karennicolejohnson.com/wp-content/uploads/2012/11/KNJohnson-2012-heuristics-mnemonics.pdf*.

Tours help you focus your testing, Cem Kaner describes a tour as "a directed search through the program. Find all the capabilities. Find all the claims about the product. Find all the variables. Find all the intended benefits. Find all the ways to get from A to B. Find all the X. Or maybe not ALL, but find a bunch."[16]. With the combination of different tours in different perspectives (see the I SLICED UP FUN heuristics) coverage and test depth can be chosen.

Examples of Tours[17] include:

- **Configuration tour**: Attempt to find all the ways you can change settings in the product in a way that the application retains those settings.
- **Feature tour**: Move through the application and get familiar with all the controls and features you come across.
- **Structure tour**: Find everything you can about what comprises the physical product (code, interfaces, hardware, files, etc.).
- **Variability tour**: Look for things you can change in the application - and then you try to change them.

16 kaner.com/?p=96; also see developsense.com/blog/2009/04/of-testing-tours-and-dashboards/

17 from michaeldkelly.com/blog/2005/9/20/touring-heuristic.html

Personas

Personas can be used to reflect various users of software **who** will be expected to use the mobile app. They may be designed to reflect, or model, a specific individual or a set of key criteria for a group of users. Regardless of how they are created each persona is singular, not a group of people. Personas can be used to have a clear picture of various end-users to include so that representative tests are executed for those end-user. Various research material are available at *personas.dk*.

Testing on Various Devices

Some bugs are universal and can be discovered on any mobile device. Others, and there are plenty of them, are only happening on a subset of devices or in a certain environment (see paragraph above: Beware Of Differences). Rather than trying the never-ending task of a full testlab another solution is possible: start testing in production, for instance by involving end-users who use their own devices and combinations.

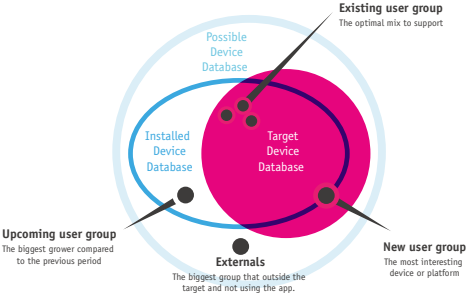
Physical and Virtual Devices

Physical devices are real, you can hold them in your hands. Virtual devices run as software, inside another computer. Both are useful hosts for testing mobile apps.

Virtual devices are generally free and immediately available to install and use. Some platforms, including Android, allow you to create custom devices, for instance with a new screen resolution, which you can use for testing your apps even before suitable hardware is available. They can provide rough-and-ready testing of your applications. Key differences include: performance, security, and how we interact with them compared to physical devices. These differences may affect

the validity of some test results. Beside the Android platform virtual devices you can use *GenyMotion.com*, a faster and more capable Android emulator, for instance, to control sensor values.

The set of test devices to use needs to be reviewed on an ongoing basis as the app and the ecosystem evolve. Also, you may identify new devices, that your app currently does not support, during your reviews. The following figure illustrates these concepts.



Ultimately your software needs to run on real, physical, phones, as used by your intended users. The performance characteristics of various phone models vary tremendously from virtual devices on your computer. So: buy, rent, beg, borrow phones to test on. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard, screen resolution, networking chipset, et cetera. Try your software on at least one low-end or old device as you want users with these devices to be happy too.

Here are several aspects to test explicitly on physical devices as the devices have a significant impact on these aspects:

- **Navigating the UI:** for instance, can users use your application with one hand? Effects of different lighting conditions: the experience of the user interface can differ in real sunlight when you are out and about. It is a mobile device – most users will be on the move. Rotate the screen and make sure the app is equally attractive and functional.
- **Location:** if you use location information within your app: move – both quickly and slowly. Go to locations with patchy network and GPS coverage to see how your app behaves.
- **Multimedia:** support for audio, video playback and recording facilities can differ dramatically between devices and their respective emulators.
- **Internet connectivity:** establishing an internet connection can take an incredible amount of time. Connection delay and bandwidth depend on the network, its current strength and the number of simultaneous connections. Test the effects of intermittent connectivity and how the app responds.

As mentioned earlier, crowdtesting can also help to cover a wide range of real devices, but you should never trust on external peoples' observations alone.

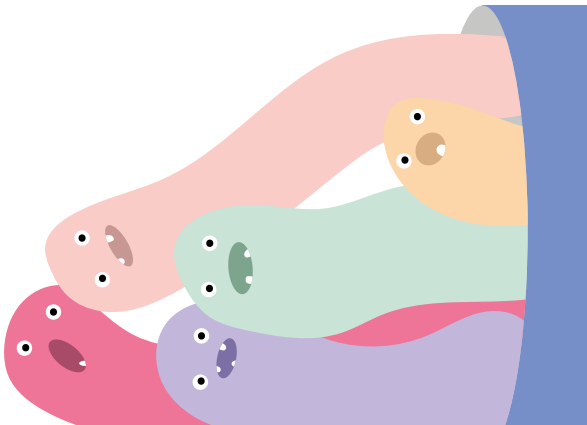
Remote Devices

If you do not have physical devices at hand or if you need to test your application on other networks, especially abroad and for other locales, then one of the 'remote device services' might help you. They can help extend the breadth and depth of your testing at little or no cost. Device farms are becoming commonplace, and are clearly strategic where Google and Amazon in particular now provide them.

You can also use commercial services of companies such as *SauceLabs.com*, *Testdroid.com*, *PerfectoMobile.com* or *Sigos.com* for similar testing across a range of devices and platforms. Some manufacturers brand and promote these services however you often have to pay for them after a short trial period. Some of the commercial services provide APIs to enable you to create automated tests.

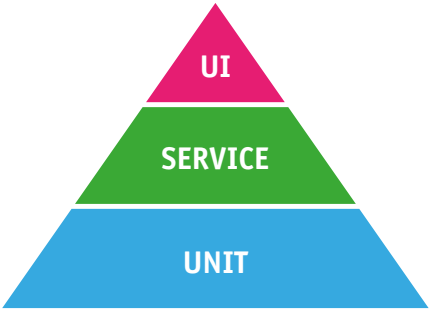
You can even create a private repository of remote devices, e.g. by hosting them in remote offices and locations.

Beware of privacy and confidentiality when using shared devices.



Test Automation

Automated tests can help you maintain and improve your velocity, your speed of delivering features, by providing early feedback of problems. To do so, they need to be well-designed and implemented. Good automated tests mimic good software development practices, for instance using Design Patterns[18], modularity, performing code reviews, et cetera. To automate scripting and coding skills are needed. The level of skills is dependent on the chosen tool. Test automation tools provided as part of the development SDK are worth considering. They are generally free, inherently available for the particular platform, and are supported by massive companies. Test automation can be performed at different levels, see the automation pyramid figure below. It is a strategic choice what should be automated in the unit tests, what on the service or API level and what scenarios on the UI level of the application. The pyramid represents trust that is built up from the unit test to the higher levels. Multiple test levels are needed to prove that the app works.



18 en.wikipedia.org/wiki/Design_Patterns

GUI Level Test Automation

The first level of automation are the tests that interact with the app via the Graphical User Interface (GUI). It is one of the elixirs of the testing industry, many have tried but few have succeeded. One of the main reasons why GUI test automation is so challenging is that the User Interface is subject to significant changes which may break the way automated tests interact with the app.

For the tests to be effective in the longer term, and as the app changes, developers need to design, implement and support the labels and other hooks used by the automated GUI tests. Both Apple, with their XCTest framework[19], and Android[20] underpin their test automation frameworks with Accessibility features incorporated into their platforms.

Some commercial companies have open sourced their tools, e.g. SauceLabs' appium[21] and Xamarin's Calabash[22]. These tools aim to provide cross-platform support, particularly for Android and iOS. Other successful open source frameworks include Robotium[23] which now offers a commercial product - a test recorder. Several other tools have effectively disappeared, perhaps the industry is now maturing where only the stronger offerings survive?

19 developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html

20 developer.android.com/training/testing/ui-testing/uiautomator-testing.html

21 https://github.com/appium/appium

22 github.com/calabash

23 github.com/robotiumtech/robotium

Service Level Test Automation

There is a lot of business logic implemented inside an API. Changes in this logic or in the backend system can be monitored with automated API tests. The focus of the test can be functional-regression but also reliability, performance and security. For functional regression testing a tool like Postman[24] is useful.

Several tools can help with API testing. They include Fiddler by Telerik[25] and Charles Proxy[26]. Both enable you to view and modify network traffic between your mobile device and the network.

Unit Level Test Automation

Unit testing involves writing automated tests that test small chunks of code, typically only a few lines of source code. Generally, they should be written by the same developer who writes the source code for the app as they reflect how those individual chunks are expected to behave. Unit tests have a long pedigree in software development, where JUnit[27] has spawned similar frameworks for virtually all of the programming languages used to develop mobile apps.

24 getpostman.com

25 telerik.com/fiddler

26 charlesproxy.com

27 en.wikipedia.org/wiki/JUnit

BDD Test Automation

BDD is a Behavior-Driven Development[28] approach within the Test Driven Development family. The behavior is described in formatted text files that can be run as automated tests. The format of the tests are intended to be readable and understandable by anyone involved with the software project. They can be written in virtually any human language, for instance Japanese[29], and they use a consistent, simple structure with statements such as **Given, When, Then** to structure the test scripts.

The primary BDD framework to test mobile apps is Calabash for Android and iOS[30]. Various others have not been developed or maintained in the last year and can be considered defunct for all but the most persistent developers. General purpose BDD frameworks may still be relevant when they are integrated with frameworks, such as Appium, that use the WebDriver wire protocol (a W3 standard)[31].

Automation can also help the manual testing, for instance, to replace manual, error-prone steps when testing, or to reduce the time and effort needed, for instance, to automate a collection of screenshots. Developers can help testers to be more efficient by providing automated tools, e.g. app deployment via ADB[32].

28 en.wikipedia.org/wiki/Behavior-driven_development

29 github.com/cucumber/cucumber/tree/master/examples/i18n/ja

30 github.com/calabash

31 w3.org/TR/webdriver

32 thefriendlytester.co.uk/2015/11/deploying-to-multiple-android-devices.html

Testing Through The Five Phases of an App's Lifecycle

The complete lifecycle of developing a mobile app fits into 5 phases: implementation, verification, launch, engagement and validation. Depending on which phase(s) you are involved in, there are different tasks for a mobile app tester to be performed. For example, when joining a development team, the task can be the analysis of the error in the log files on a device. When joining a beta test phase, a task can be the analysis of usability tests results like recording movies.

Testing applies to each phase. Some of the decisions made for earlier stages can affect your testing in later stages. For instance, if you decide you want automated system tests in the first phase they will be easier to implement in subsequent phases. The five phases might suggest that they follow one after the other; this is not the case. Every step in the different phases provides the possibility to learn and improve. When testing the team learns both how good the mobile app product is and also about areas for improvement in how the app is produced. Mobile app development is a challenging complex, dynamic activity that does not go perfectly the first time, therefore, teams should incorporate an improvement cycle so they can learn and actively improve what they do, see Mobile improvement model[33].

Phase 1: Implementation

This includes design, code, unit tests, and build tasks. Traditionally testers are not involved in these tasks; however good testing here can materially improve the quality and success of

33 https://improvement.polteq.com/en/ti4mobile/

the app by helping us to make sure the implementation is done well.

In terms of testing, you should decide the following questions:

- Do you use test-driven development (TDD)?
- Help review designs on what are the main, alternative and negative user flows
- Which test data do you use to validate the user flows?
- Will you have automated system tests? If so, how will you facilitate these automated system tests? For instance by adding suitable labels to key objects in the UI.
- How will you validate your apps? For instance, through the use of Mobile Analytics? Crash reporting? Feedback from users?

Question the design. You want to make sure it fulfills the intended purposes; you also want to avoid making serious mistakes. Phillip Armour's paper on five orders of ignorance[34] is a great resource to help structure your approach. And again: do read the first chapters in this guide to learn more about this important phase in app development.

Phase 2: Verification

Review your unit, internal installation, and system tests and assess their potency: Are they really useful and trustworthy? Note: they should also be reviewed as part of the implementation phase, however, this is a good time to address material shortcomings before the development is considered 'complete' for the current code base.

34 www-plan.cs.colorado.edu/diwan/3308-07/p17-armour.pdf

For apps that need installing, you need ways to deploy them to specific devices for pre-release testing. Based on your test strategy you can decide on which phones, platforms, versions, resolutions are in scope of testing and support.

System tests are often performed interactively, by testers. You also want to consider how to make sure the app meets:

- Usability, user experience and aesthetics requirements
- Performance, particularly as perceived by end users[35]
- Internationalization and localization testing

Phase 3: Launch

For those of you who have yet to work with major app stores be prepared for a challenging experience where most aspects are outside your control, including the timescales for approval of your app. Also, on some app stores, you are unable to revert a new release. So if your current release has major flaws you have to create a new release that fixes the flaws, then wait until it has been approved by the app store, before your users can receive a working version of your app.

Given these constraints, it is worth extending your testing to include pre-publication checks and beta tests of the app such as whether it is suitable for the set of targeted devices and end-users. The providers of the main platforms publish guidelines to help you test your app will meet their submission criteria. These guidelines may help you even if you target other app stores and can be used as a checklist during the implementation phase.

35 A relevant performance testing tool is ARO (Application Resource Optimizer) by AT&T: developer.att.com/application-resource-optimizer, open source project at github.com/attdevsupport/ARO

App Store Guidelines	
Apple	developer.apple.com/app-store/review/guidelines/
Android	developer.android.com/develop/quality-guidelines/

Phase 4: Engagement

This includes search, trust, download and installation. Once your app is publicly available users need to find, trust, download and install it. You can test each aspect of this phase in before and in production. Try searching for your app on the relevant app store, and in mainstream search engines. On how many different ways can it be found by your target users? What about users outside the target groups - do you want them to find it? How will users trust your app sufficiently to download and try it? Does your app really need so many permissions? How large is the download, and how practical is it to download over the mobile network? Will it fit on the user's phone, particularly if there is little free storage available on their device? And does the app install correctly? - there may be signing issues which cause the app to be rejected by some phones.

Phase 5: Validation

This includes payment, usage and user feedback. As you may already know, a mobile app with poor feedback is unlikely to succeed. Furthermore, many apps have a very short active life on a user's phone. If the app does not please and engage them within a few minutes it is likely to be discarded or ignored. And for those of you who are seeking payment, it is worth testing the various forms of payment, especially for in-app payments.

Consider finding ways to test the following as soon as practical:

- Problem detection and reporting. These may include your own code, third-party utilities, and online services.
- Mobile Analytics. Does the data being collected make sense? What anomalies are there in the reported data? et cetera?
- Feedback. For all the flaws and limitations we can glean a lot from feedback users provide including their sentiments, feature requests, bugs, and clues we can use to improve our testing.

You can read more about Mobile Analytics and Feedback in this book.

Learn More

Testing mobile apps is becoming mainstream with various good sources of information. Useful online sources include:

- *katrinatester.blogspot.de/2015/08/mobile-testing-pathway.html* - A comprehensive and well presented set of possible steps for testing mobile software.
- *github.com/julianharty/testing-heuristics* - An online open source project to learn more about testing heuristics for mobile apps.
- *enjoytesting.files.wordpress.com/2013/10/mobile_testing_ready_reckoner.pdf* - Contains short, clear testing ideas with examples, mainly for Android devices
- *developers.google.com/google-test-automation-conference/* - The annual Google Test Automation Conference (GTAC) often includes several presentations on testing mobile

apps. These are recorded and available free of charge, worth watching.

- *testdroid.com/blog*- A fertile blog on various topics including testing mobile apps. They also have a series on testing mobile games[36].
- *genymotion.com/blog/android-testing-showdown/* - A useful guide on selecting the best devices to test on.
- *appqualityalliance.org/resources* - The official App Quality Alliance AQuA website including their useful app testing guidelines.

A good place to start learn testing mobile apps is reading books like:

- *sensible.com/* - Steve Krug has written several immensely popular books that cover low-cost d-i-y usability testing. Various chapters are available online, his work is well worth reading and much applies to testing mobile apps.
- *leanpub.com/testmobileapps* - Tap Into Mobile Application Testing, by Jonathan Kohl provides help and advice on ways to test mobile apps.
- *handsonmobileapptesting.com/* - Hands-on Mobile App Testing, by Daniel Knott. A well-written book on various aspects of testing mobile apps. A sample chapter is available from the web site.

And of course do not miss the platform-specific articles in this guide, especially the mobile web chapter to gain deeper insights into mobile testing.

36 testdroid.com/testdroid/7790/best-practices-in-mobile-game-testing