TAP INTO MOBILE APPLICATION TESTING

JONATHAN KOHL

Tap Into Mobile Application Testing

Jonathan Kohl

This book is for sale at http://leanpub.com/testmobileapps

This version was published on 2013-09-14

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Jonathan Kohl by spreading the word about this book on Twitter!

The suggested hashtag for this book is #testmobileapps.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#testmobileapps

For my wife Elizabeth. Thanks for your support and pushing me to write this book in the first place.

Contents

Chapter 1: Get Started Now

> "How do you start testing software? You just start using it." — Javan Gargus

If you're new to testing, or to testing on mobile devices, it can be a bit nerve wracking at first. You may have questions like:

- What do I *do*?
- What information do I report?
- What if I don't find any problems?

Don't worry. You'll learn plenty of approaches to help you answer each of these questions as you work through this book. Especially don't worry about the third question. If you follow along and try some of the approaches I write about, you *will* find problems.

When I first started out in software development, my colleague Javan and I worked closely together as testers and became good friends. At a conference, we met a customer of the software we had tested. She remarked on how robust and reliable the software was, that we must be highly

skilled testers and that she appreciated our work. She used other software from other vendors that would crash, perform poorly, freeze up, and was difficult to use.

She was interested in what we did differently and how we started new projects. Javan shrugged and said, "We just start using it."

Javan had a good point, but testers do more than use the software. They use it systematically, observe carefully and evaluate bravely. Expanding on *how* testers use it, you must:

- Use the software systematically,
- Observe what is going on carefully,
- Evaluate its effectiveness of it bravely,
- Investigate anything that piques your curiosity,
- Record and report anything interesting.

Still concerned about whether you'll find bugs? Don't be. If you're systematic and observant, you *will* find important problems.

The difference between a great tester like Javan and testers who provide little or no value to a project is in how the great tester uses the software, what he pays attention to, how he evaluates it, and what he reports. To help you to be a great mobile tester, much of this book is dedicated to addressing these concepts. Here are some of the approaches Javan and I used.

Gather User Information

To start off, we found as much information about our end users as we possibly could. Then, we tried to use the software the way they would and we evaluated the software to gather the following kinds of information:

- What goals or tasks are customers trying to achieve with this software?
- How do customers use computers and other software?
- Under what conditions do customers use the software?
- What happens when something goes wrong?

If you're missing some of this information, you can still rely on one user's information: *yours*. That's right, you. You are a user of software, and if you are reading this, then you probably have an idea of what mobile devices are. If you don't have much information to go on, then use what you know or don't know.

While we are all unique, we tend to behave in similar ways. (We all may be unique, individual snowflakes, but we still act like snowflakes.) That means there are people just like you who will use the software you're testing in much the same way. They'll get confused about the same things you get confused about. They'll struggle with the same parts

of the app that you struggle with. They'll enjoy the same features and aspects that you enjoy.

So, if you can't get all the information I mention above—or at least not right now (please try to get it at some point while you test)—then just start using the software. Be aware and systematic. Pay attention, and watch for anything out of the ordinary.

Start Testing

Now, grab your mobile device and start testing. Do you have an app that requires your attention? Or maybe you're just learning about this topic and you don't have a programmer or team depending on you. If that's the case, take your device and pick an app—any app, even something built in like the camera or maps app—and start using it.

As you use it, think like an investigator. Note any interesting information. Grab paper and pen, or use a note-taking app to record what you see. Try answering the following questions:

- What are your first impressions?
- Is anything confusing?
- Does the app feel slow?
- Where are you testing it?
- What hardware device, OS version and network type are you using?

- What's the weather like? (No, this isn't a joke. You'll learn why in the next chapter.)
- Does the app crash or freeze? (If so, note the specific steps you need to take to repeat that behavior.)

What Are "Bugs" Anyway?

We tend to consider crashes, app freeze-ups wrong answers, and incorrect calculations as bugs. These are the issues that programmers are interested in right now. However, any of the other questions above can lead to discovering important bugs if you take the time to investigate.

I get my definition of a bug from James Bach: "A bug is something that bugs someone who matters." If it bugs you, then log it. *You*, as a tester, are someone who matters.

Reporting *qualitative* information is especially important. If an app is confusing or frustrating, it may have serious design flaws that need to be addressed. Mobile apps, more so than any other kind of app, depend on usability and performance. Why? Because that is what consumers expect and demand. If an app is hard to use or if it is too slow, it will get deleted from devices and a user will move on to a competing app.

The choice to delete an app from a device is usually a qualitative one. It may be functionally correct—that is, it meets a specification—but how the user *feels* about the app is important. Mobile apps are easy to install and even

easier to delete. That's why it's important to note how you feel when you are using the app. Those feelings have an enormous influence on whether an app is used or not.

I want you to remember the following:

NEVER, UNDER ANY CIRCUMSTANCES, BLAME YOURSELF FOR FEELING CONFUSED BY THE TECHNOLOGY!

Phew. Sorry for shouting. Let me repeat that a little more calmly:

Never, under any circumstances, blame yourself for feeling confused by the technology!

Got it?

Wait, what's that? You don't believe me?

Mobile devices aren't easy to use. They are smaller than non-mobile devices, harder to type with, and we use them in all sorts of weird situations and locations when we are on the move. If an app is confusing or hard to use, people get frustrated because they don't have the luxury of comfort during use. If *you* find an app hard to use, you can bet that others will, too.

Technologists spend a lot of time, money and effort making applications work well. Always remember: Technology exists to serve you, not the other way around. Technology should help you do a job, entertain or delight you. It should not make you feel stupid. If it doesn't help you, or if makes you feel stupid or inadequate—yes, I mean you, the one

reading this sentence—then there is a serious problem with the software that needs to be reported right away.

First Launch Test

OK, here is something hands on. I want you to try this on your mobile app of choice. Take your time, be thorough and note anything interesting along the way:

1. Locate the icon for the app on your device home or apps screen. Can you tell what the app does and what problems it might solve by reading the title and looking at the icon picture?
2. Tap the icon to start the app. Does it load quickly, or does it sit there for a while before you can do anything?
3. Examine the splash screen as the application loads. Does it give you an idea of what the app will do? Does it create a positive image of the company? Is the load time reasonable, or does it seem to take too long?
4. Once the app loads, stop and look at it carefully.

Here are some things to observe and evaluate. Does the app ask your permission if it needs to use location services or other privacy features? Is the design clean and uncluttered or overly busy and confusing? Can you easily tell what components you can interact with (tap on links, buttons,

etc.) or are you left wondering what to do next? Is it obvious how to enable features of the app? Can you quickly and easily use the app for the purpose it was intended for, or are you left wondering?

You do all of these things when you first launch an app, but a lot of those actions are processed by your brain in your subconscious. Because of that, you don't observe closely, and you don't evaluate unless something goes very wrong.

The difference between merely using an app and using it with a systematic approach is in how you pay attention to everything you do and evaluate what the app does. And, don't forget to note the good things, the bad things and any questions or concerns you have while you're doing it!

I'm going to give you more thinking tools throughout the book to help you be more systematic in your test approach. Practice this style of usage, observation and evaluation in other areas. Try out different features, and break them down into components as I did above. If you want to try it right now, then take a break from reading, think of other systematic approaches and work through them.

Watch for Deletable Offenses!

Now, add some user information to your evaluation. Mobile end users don't have a lot of patience with apps. If something bothers them, they'll just delete the app and move on. I call this a "deletable offense." Watch for any emotion that makes you feel frustrated enough to want to

delete the app, and note what occurred. This is a crucial aspect to the success of a mobile app and a useful tool to help us find all kinds of important bugs.

When installing an app on their devices, most people tap it to start the program and, within seconds, decide whether they are going to keep it. If an app works well, they keep it. If it frustrates them, makes them angry, or doesn't meet their needs, they delete it. They make the decision quickly, and deleting an app takes about two seconds.

Here are some examples from my own experience:

- Launched the app. All the icons for local services were shown on a zoomed-out map of North America (not even a local city). When I tried to zoom in, I inadvertently tapped the icons, which took me to various service pages—usually the ones on top, not the ones underneath. The icons on top were not the services I wanted to check out. After two minutes of trying the app, I got frustrated and deleted it.
- Launched the app. It took 30 seconds to load. When it did load, I was shown a confusing home screen. Whenever I interacted with it, the app would take forever to do something. Eventually, I would realize I'd tapped the wrong thing and wanted to go back, but I had to wait. This app was too slow to be useful. Deleted.
- Launched the app. After the initial splash screen disappeared, I saw a login screen. If I didn't have an

> account, I would need to create one to use the app. I don't know this vendor very well and don't trust them enough to sign up. Furthermore, their sign-up form had a lot of fields to enter information into. It would take forever to type all that on the device. Deleted.

Here are some reasons why I will delete an app within seconds of installing it:

- The app is difficult to use.
- The app has poor performance. It's too slow.
- The app is unreliable, crashes, freezes up or is inaccurate.
- The app is lame. It has a poor visual design and execution.
- I am forced into sharing private, personal information with an organization I don't yet know or trust.
- The app doesn't work as advertised.
- It's a copy cat. The app doesn't provide me with any value over other mobile apps I already have (e.g., maps, location, web content, etc.).

As you test, listen to your emotions and watch for deletable offenses. Once you notice a negative reaction, try to identify exactly what is bothering you. From that information, you will find bugs that are important to your end users.

A Tester's Mindset

Many users blame themselves for errors that occur when using technology, thinking that maybe they did something wrong. You must reverse this belief if you want to be an effective tester. Here is a rule of thumb: If something unexpected occurs, don't blame yourself; *blame the technology*.

Former Apple vice president Donald Norman's classic book *The Design of Everyday Things* demonstrates how many things around us are poorly designed, counterintuitive and leave us feeling silly. If you read it, you'll never look at a door or stovetop the same way again. You'll realize the problem isn't *you*. It's just that a lot of products with poor design have conditioned you to expect mediocrity and make you feel dumb. That's not what we design technology for, and the worst response when using mobile app is a negative emotion.

Reporting Issues

When starting out testing, there are two simple categories of problems you must report:

1. A clear program malfunction occurs (the app crashes, freezes, ruins data, provides incorrect results, etc.).
2. Something bothers me (annoyances, the app or workflows within it are awkward, poor performance, lack of feedback, etc.).

Keep notes of anything related to those categories, even if it's only a rough record. You can always go back and find out more information later. Add exact steps and what you think should have happened instead so that you can log the issues in a way that others can follow and reproduce. A bug report that can be followed, understood and reproduced can result in a fix.

Speaking of details, make sure that you test any statement about the software and provide evidence to back it up or contradict it. Never assume anything just because someone (a programmer, coworker, manager, designer, etc.) says it is so. One of my rookie mistakes was to focus only on areas the programmers told me to focus on and to avoid areas they told me to avoid. However, when others—especially our customers—found bugs in areas I had avoided, *I* was on the hook.

If you are asked how well you tested a certain feature or area of a program and you sputter out something about not doing much because the programmer asked you not to, it sounds foolish. After all, if you're just going to test what the programmer asks you to test, how useful are you to the team?

Your job is to gather data and observe whether assumptions about the product hold up. If someone says that they aren't worried about a certain area of an app and not to test it, then you had better make sure that there is evidence to back this up—even if you wait to test it until after you test the areas they told you to focus on. A little evidence to the

contrary goes a long way to change people's perceptions about the quality of an app.

As testers, our job isn't to assume. It's to prove ideas by testing. Our standard is to use cold, hard evidence to measure those ideas. We believe in facts, not assertions. We seek proof, so we need to have courage to think for ourselves (even if you're new at this).

Don't Be a Jerk

A word of caution: Once you get used to this kind of questioning mindset (like that of a scientist or professional investigator), don't become a contrary jerk. There's nothing worse than a team member who thinks he or she is a last line of defense between the programmers (or the rest of the team) and the customers. These kinds of people seem smug, irritating and unapproachable. They lose credibility because they're hard to get along with. That means their information may get ignored or treated with less respect due to their abrasiveness.

Have courage and prove things out, but trust that other team members are also trying to do the right thing. When we test, it's our job to provide evidence to our team's assertions and ideas, not to be the police force. Remember, we're all working together to create an amazing product. Some of us just might have different ideas about what that should look like.

When you test out an assertion that proves to be wrong

(e.g., someone tells you not to worry about testing a feature, but you spend a bit of time on it anyway and find a major bug), do not run around saying, "I told you so!" Report it in a neutral fashion, and make sure you have evidence—exact steps to reproduce the problem under credible conditions—and let that evidence do the talking.

Other team members may not be happy with that evidence in the moment, but when the product is better and your customers are happier, they will love you for it. Eventually.

Improving Your Approach

When I was a rookie tester, I didn't have a lot of guidance, but I was fortunate to work on a software development team that provided a lot of support and access to other team members. At first, my testing looked like this:

1. Try to use the software by either figuring it out myself or looking at user guides.
2. Ask programmers to tell me what they would like me to test.
3. Ask team members if something that bothered me was a bug or not.
4. When needed, ask programmers to show me how to test something I didn't understand.
5. Review my findings and ask others if I should formally record problems as bugs or not.

Notice my dependence on others for most of my testing work. What do you think happened?

If you guessed the following, pat yourself on the back:

- My testing results were inconsistent.
- Some problems that others told me not to record as bugs were later discovered and reported by our customers.

In short, I missed discovering important problems because either I completely missed a problem someone else discovered or I talked myself out of logging a bug—or let someone else talk me out of it—only to have it reappear later.

If I missed an important bug or, even worse, discovered it and didn't log it, that called my credibility and ability into question.

To improve, I started to take personal responsibility of my work and stopped relying on others to guide my testing and problem reporting. It required both research into how to determine testing for a project and a mindset change.

Testing Coverage

When I wanted to improve my testing approach, one of the first people whose work I studied was Cem Kaner. In his paper "Software Negligence and Testing Coverage[1]," Cem

[1] http://www.badsoftware.com/coverage.htm

describes 101 different models of coverage. From Cem's work, I learned about different ways of determining testing coverage and that coverage involves testing an application using a particular approach or perspective.

Many people describe testing coverage as a singular thing, but Cem showed me how you could look at the same program or product in many different ways. The more perspectives you use, the more information you can gather.

Changing your testing perspective is a powerful tool. It's amazing how much more you observe when you use an application, change your perspective and focus, and use it again. You see things you previously missed. By combining different models of coverage and changing your perspective in several different ways, you'll find much more important information in a shorter period of time.

Some coverage models are related to testing techniques or approaches. There are a lot of them. Here are some examples:

- Functional (verify specs, requirements)
- Data (app can handle and process different types of data, whether typed in, utilized through an information service or from other programs or files)
- Regression (repeating tests)
- Performance (app is quick and responsive)
- Localization (app can handle different languages)
- User scenarios (create credible usage stories and follow them)

- Usability (have real end users try to complete tasks with your app, and observe areas they struggle with)

You can define coverage in many different ways—using different test techniques, tools or scenarios, or simply testing the app in different environments. In later chapters of the book, you'll find several different perspectives you can use.

Identifying Problems

Always remember Bach's definition of a bug: *A bug is something that bugs someone who matters.* This is my rule of thumb for determining whether something is a "bug" or something else. If it bugs me, I report it.

Software testing is a simple concept, but it can be framed in many different ways. As we saw earlier, testing often involves using the software, observing what happens, generating test ideas, adapting your activities, and reporting behavior that team members might be interested in.

Arguably, the most important things to note are problems, generally referred to as "bugs." When we test, the rest of the team members usually want us to find important problems quickly or demonstrate the absence of problems to determine whether or not a product is suitable for use. Your colleagues depend on you and your ability to identify problems.

Don't worry too much about whether what you observe is a bug or not. You'll figure that out as you work with a team.

Just note and report on anything that bugs you, even if it is just a qualitative impression, not a bug report—"This is great! I love the app" or "Something feels wrong when I do this." Sometimes, this information can be even more useful than a formal bug report, because it helps the team get feedback on usability. As I've said before—and will again, many, many times—usability is one of the most important factors for mobile apps.

Example Problems

Here are some examples of common problems you might encounter with mobile applications.

The Program Stopped Working

This is also commonly referred to as a "crash." If the program quits unexpectedly while you are using it, that's a big problem that you need to report right away. There are various ways this occurs:

1. The program just disappears. It was there, but now it's gone.
2. The app stops working suddenly but displays an error message informing you that something went wrong.
3. It "hangs"—i.e., it just stops responding. You have to shut it down using the device's operating system.

Wrong Answer

The application is wrong in calculations, information, or location. It shows a price that is wrong, date and time information is incorrect, or it has adjusted and the device seems to be mixed up. It provides the wrong output, gets your location wrong, or can't decide what to display.

Strange Behavior

Mobile devices depend on a lot of different states: constant movement of the person and the device, determining locations while moving, and wireless communication conditions, to name a few. There are so many combinations of things that can be going on at any given time, an app may behave strangely while you are using it. Inadvertent movement, poor wireless and even different weather and light can cause things to go wrong. Watch for:

1. Screen redraw issues—After user input or after moving the device around, the screen is all garbled, has different sized objects displayed or just plain-old looks weird.
2. Odd error messages—You perform an action, and a message pops up that doesn't make sense, often filled with symbols and words that don't help you fix the issue and continue on.
3. Strange delays or pauses that interrupt your flow of thought and what you are trying to accomplish.

Information Corruption and Deletion

If something goes wrong in an app, information you entered might get messed up and become wrong, unreadable, or inaccurate, or removed altogether. Watch for the following kinds of problems when you are working on a mobile app, related to data and information.

1. Your saved work or personal settings get deleted next time you open the app.
2. Only part of the content shows up, or a screen only partially loads.
3. Something you saved in the past is missing information, or some of the information is wrong, unreadable or garbled.
4. The application crashed and caused corruption or deletion as a result.

Usability Issues

Usability and user experience are large, related disciplines. If you want to learn more, research both of them by searching with your favorite web search tool. Usability and usability testing can provide a lot of ideas for test execution, as well as guidelines. User experience branches out into different areas, such as performance and context.

Something Happened and I Don't Like it

This is a very broad category, but listen to your emotions to determine whether there is a problem or not. Do you feel frustrated? Tired? Angry? These are clues that the software is behaving incorrectly. Try to figure out exactly what is causing you to feel that way. It might be a combination of things.

Objectionable Look and Feel

Is the design of the app confusing? Is it garish? Does it look OK under different lights? Does it violate development guidelines for the platform the app was developed for? Are items too numerous or too small to see? Does it look cohesive and smooth or cluttered and thrown together?

Long Workflows

It takes extra effort to interact with and enter information into mobile apps. Typing on touch screens is especially challenging, particularly when you are on the move. Does it take forever to achieve a goal in the app? If it does, then that workflow needs to be simplified or it will frustrate users. They may even feel that the performance of the app is poor just because it takes so long to do something.

Unclear Wording and Language

There isn't a lot of space for words and text in mobile apps, so we have to choose our language carefully. Do the words

help or confuse? Are they vague or specific? If they are vague, how many ways could they be interpreted? What happens if you try to misinterpret on purpose?

For apps that need to support different languages, this is even more important to test in every language. Translation is not an exact science, and it is easy to get the context wrong and have a technically correct translation that is humorous, nonsensical or even offensive.

No Undo or Go Back

How many times have you accidentally hit the "Emergency Call" button on your smartphone and then wildly swiped at it to make sure it cancelled?

If you're like me, this probably happens several times a week. It's really easy to accidentally navigate somewhere you don't mean to go with a mobile app because they move around and get jostled. This can trigger events due to unintentional touch-screen interaction, accidentally hitting buttons, or setting off movement or other sensors.

Within an app, you may go down a path, change your mind and want to reset or start over from the home screen. Many apps don't support this very well.

When to Get Started

Sometimes people ask me when to get involved on a project and start testing. I have a mantra:

Test early, test often and test in the real world.

As soon as there is something to evaluate and provide feedback on, start testing. That's usually right at the beginning of the project.

There are a couple of good reasons for this. First, time is at a premium on mobile projects. If you wait until a polished product build is available near the end of the project, you won't have much time at all to test it. Second, it's much easier and often cheaper to fix problems earlier on a project than later on. Also, great testing feedback can help shape a product so that it is more usable and can have better performance.

Introducing Tracy Lewis

Tracy is a friend of mine; her husband David McFadzean and I have worked on a number of projects together. It must have sounded fun, because last summer she asked if I could train her to become a mobile tester. I agreed to mentor her on one condition: she needed to review this book. If she could use it to get started and be effective as a mobile tester, then I had done my job.

Tracy Lewis (photo by Todd Kuipers)

To provide you with an alternate voice, Tracy will share her thoughts on topics covered in each chapter. Watch for asides just like this one to read her ideas and perspective:

Tracy's Thoughts

Tracy: This chapter helped me gain confidence to test. When you said to never blame yourself for feeling confused about technology, that really helped build my confidence. It's not *my* problem, it's a problem with the technology! Also, if it bugs me, then it's a bug worth logging.

Reading *The Design of Everyday Things* emphasized when I run into something awkward, it isn't just me. Lots of people struggle with the design of doors, or household appliances, and any time I struggle, I need to note that and communicate that back to the team.

It was also interesting to read about not being a jerk. I could see how it could be easy to criticize and forget that there are real people who have worked really hard on this project. What we say matters, and we need to be careful and professional in our communication. If we get along well and are respectful, that makes for a happier team.

Concluding Thoughts

To get started testing, just use the app in a systematic way. Carefully observe what is going on, evaluate what you see

and record everything that might be useful for others to know.

Try to think like a scientist or an investigator. You are not driven by emotion, and you are a smart, capable tester with natural skills and emotions. Follow your instinct, and *always* get data and evidence to back up any assertion that you or your teammates make about the software.

If the app makes you feel confused, stupid or unsure of yourself, it is the app's fault. Try to figure out why it is doing that, even if you just explain or demonstrate what is happening and what you are thinking to a colleague who might see something you are missing and may be able to help you express it better. No problem or issue is too trivial. Report them all. Let other people triage the issues and decide what to do with them. Don't self-censor. It is better to have too much information than too little.