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

Acknowledgements i

Introduction to the Book ii

Chapter 1: Get Started Now 1

Chapter 2: Explore Mobile Technology 27

Chapter 3: Understanding Wireless Technology . 96

Chapter 4: Mobile Testing Tours 112

Chapter 5: Test Using Different Perspectives . . . 159

Chapter 6: Logging and Diagnosing Bugs 234

Chapter 7: Mobile Test Strategy 277

Chapter 8: Guidance and Planning 314

Chapter 9: Test the Mobile Web 390

Chapter 10: Don't Forget about Performance and Security! **465**

Chapter 11: Final Thoughts **491**

Appendix 1: Tools and Considerations **500**

Appendix 2: Mobile Content Snack Tests **509**

About the Author **512**

Acknowledgements

Thanks to Tracy Lewis for being my protege on this project. Your hard work, willingness to learn and attention to detail helped me shape this content.

Thank you Cathy Clinger for the fabulous cover design.

Thanks to Joey McAllister (the best damn copy editor on the planet) for your help and the constant encouragement.

Thanks to my reviewers: Johan Hoberg, Tracy Lewis, John Carpenter, Elizabeth Lam, David McFadzean, Martin Jansson, Chris Garrett, Javan Gargus, Jared Quinert, Aaron West and David Greenlees. Your contributions and encouragement to keep going helped me take this from a rough set of ideas to a book.

Thank you to Cem Kaner and James Bach for showing me that there was another way to test. You helped me discover other perspectives and ideas that have shaped my approach to software testing over the past 15 years. Thank you to Brian Marick for kickstarting my writing career and for encouraging me to share my ideas publicly.

Introduction to the Book

Introduction

Mobile devices are enmeshed in our everyday lives. Mobile applications, commonly called "apps," have grown from simple communication functions to gaming, entertainment and a whole host of services that help us in our everyday lives. Mobile apps are even popping up in medical and pharmaceutical industries, aviation and other business-critical areas. As we grow more dependant on them to do more important things, the need for effective testing becomes critical.

What You Will Learn

1. How mobile devices work: features, affordances and how to effectively use them when testing.
2. How to use thinking tools to provide structure to your testing and test idea generation.
3. How to recognize common failures on devices.
4. How to log mobile bugs well.
5. How to create a test strategy that makes the most use of your time and resources.

6. That it is important to incorporate physical movement into your testing.

Since devices are mobile (as in built for movement), you need to move the device (it's full of sensors), and to see how it handles handle different network connections and different environments and locations.

Whom This Book Is For

This book is written for anyone who needs to do mobile app testing. If you are a first time tester, a developer or other software development professional, or you run a big department with testers to direct, you will find something useful in this book. I've written this book to help you get up to speed on testing mobile apps quickly. Even if you are a veteran tester, there are ideas here that you might not have thought of.

I don't assume that you have a lot of experience with mobile devices, so we start with testing and mobile fundamentals and how to use devices for testing, then work from there. This book's based on my hands-on experience on mobile projects.

This book covers the following areas to help you in your mobile testing efforts:

1. Mobile technology and areas to focus on when testing

2. How effective use of mobile devices and features help you find important problems quickly
3. Common problems to watch out for.
4. Strategies to help focus your testing effectively.
5. Tools and techniques to kick-start your testing.

What We Cover

Some of you are approaching software testing for the first time. Others are professional testers used to traditional software Quality Assurance practices and tools. I try to keep both audiences in mind.

One of your primary goals generally to find important problems quickly. That's what the people who have engaged your testing services are looking for. They want you to find issues that they can fix before the app becomes available to end users.

My job is to give all of you the right tools so you can get productive as mobile testers in your unique environment. That means I will provide some guidance on how to approach testing to help get you started.

What Isn't Covered

I don't talk about test automation in this book. I have a couple of reasons why.

First of all, to have any hope of creating valuable test automation, you first need to understand how to manually

test mobile apps well. Simply creating rote automated functional tests doesn't really provide a lot of value for an incredibly complex device. Mobile apps are dynamic and dependent on physical hardware, networks and other systems to work well. Furthermore, devices are full of sensors, and our automation tools don't really help us re-create the kinds of tests that mimic what our end users actually do with devices.

Second of all, the automation tools are still in their infancy, and none of them do what I need them to do yet. I may work on this problem down the road, but for now I'll point you to the work of others who have done work in this area, such as Julian Harty.

I don't define everything you come across in the book, and I rarely mention specific mobile tools, because they go out of date quickly. If you have a question about a feature, tool, or technique that isn't addressed, I expect you to open up your favorite web browser and do a bit of research on your own. You'll have a richer experience that way.

How to Use This Book

This is a technical reference book, so expect to use it in a variety of ways. Some of you will want to read it cover-to-cover first, and others will just flip through until you find something that helps you solve an immediate problem.

Feel free to use this book in any way you choose.

However, you'll find it easier to have a device on hand as you read it to try out some of the ideas. I've found as a trainer that people find testing on mobile devices to be a lot of fun. You can get up and walk around, wave the devices themselves around to test sensor integration, and use them in all sorts of interesting scenarios. If you aren't having fun, move to another section and try again. Happy testing!

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.

Chapter 2: Explore Mobile Technology

"I have a friend who's an artist and has sometimes taken a view which I don't agree with very well. He'll hold up a flower and say, 'Look how beautiful it is,' and I'll agree. Then he says, 'I as an artist can see how beautiful this is, but you as a scientist take this all apart and it becomes a dull thing,' and I think that he's kind of nutty.

First of all, the beauty that he sees is available to other people and to me too, I believe. Although I may not be quite as refined aesthetically as he is … I can appreciate the beauty of a flower. At the same time, I see much more about the flower than he sees. I could imagine the cells in there, the complicated actions inside, which also have a beauty. I mean, it's not just beauty at this dimension, at one centimeter; there's also beauty at smaller dimensions, the inner structure, also the processes.

The fact that the colors in the flower evolved in order to attract insects to pollinate it is

> interesting; it means that insects can see the color. It adds a question: Does this aesthetic sense also exist in the lower forms? Why is it aesthetic? All kinds of interesting questions which the science knowledge only adds to the excitement, the mystery and the awe of a flower. It only adds. I don't understand how it subtracts." — Richard Feynman

You there! Yes, you with the mobile device! I need you to pick up your smartphone or tablet (or both) and follow along for this chapter. This is important. Understanding how the devices work will determine whether you figure out why a bug occurs five feet away from you but doesn't occur where you are sitting. (Do you know why this could occur? If not, don't worry. You should have a better idea by the end of this chapter.)

First, I want you to pick up your device and study it. Look at the front, look at the back and look at the sides. What do you see?

Now, move the device around. Wave your arm back and forth. Tilt the device, shake it, spin it in a circle and cover up the screen with your hands. Do you know what's going on when you do these things?

With PCs, not much happens when you move the device. They are built to be plugged in and stable. Mobile devices, on the other hand, are *mobile*, and they are chock-full of sensors and other services to help us work with them

and interact with them when we're on the move. In fact, they are a big component in what makes smartphones and tablets so awesome. Without sensors, location services, cameras and communication features, we'd just have really small, really lame computers in our pockets.

Sensors in particular have an enormous role in what makes mobile devices so powerful.They are amazing, tiny things that help make our mobile world go around. Much like Richard Feynmman's flower, there is more to these devices than meets the eye. Beyond the shiny, attractive case, a mobile device comprises dozens of sensors, services, software and firmware that all need to work together in harmony.

In this chapter, we'll go through some of the more popular mobile features that we need to be aware of when testing.

What's Inside the Device?

Smartphones and tablets are absolutely jam-packed with technology to be able to support the features that we enjoy. The effort that goes into the combination of materials and tech to make touchscreens that work in different lights, respond to gestures and don't smudge up until they are unusable is amazing.

If you open up a device (check out ifixit.com[2] for device teardown articles and pictures rather than doing it yourself), you'll see a lot of small physical pieces that fit

[2] http://www.ifixit.com

together. You have an integrated circuit board, a battery and maybe a memory chip from your telecommunications provider (a SIM card or something similar that gives you cellular access, stores your contacts, etc.). You'll also see items that are plugged into the board—sensors, motors, etc.—and a camera.

You may not see them, but there are also antennas to help the device connect to services that some of your apps will depend on. All of these items work together to create a mobile experience, and app developers can tap into some of them directly to enhance their mobile apps. It's important to understand what is inside the device so that you can test apps thoroughly and understand how they might fail in ways that a PC or web app will not. Go online and search for information about the device you are testing to learn about its features.

Physical Features

When testing on PCs, we don't often think of how they are physically composed, or how interacting with buttons, moving the device, or plugging and unplugging accessories might impact the software we are testing. On mobile devices, *movement* and the physical device are important to consider, because we hold them, interact with them, and the technology supports movement and mobility. Physical interaction with devices is vital because they are based on physicality, not just with movement, gesture and touch sensors, but we plug things into them, adjust buttons,

and use them in ways that can interrupt apps and other functions on the devices.

Most smartphones and tablets have the following visible features:

- Hard case (for structural support and holding the device together)
- High-resolution touch screen (for viewing and interacting with apps)
- Speakers and microphone (to listen to sounds and input sound)
- Volume control and mute buttons (to control volume and turn off the ringer)
- Buttons:
  - Navigation (to set your OS to a known state, go back, etc.)
  - Sleep/wake/lock (for when the device is not in use)
  - Some devices may also have a full keyboard
- Camera (to capture images and video)
- Headphone/speaker jack (for speakers, microphones and other accessories)
- Connector (for charging and accessories)

Now, think about what is inside. What don't you see? Here are some of those hidden components:

- Antennas (for communication, location determination, etc.):

  - Wi-Fi (network connections)
  - Global positioning satellites (GPS)
  - Bluetooth
  - NFC (near-field communication)
  - Cellular networks for data and voice (GSM, GPRS, CDMA, etc.)
- Sensors for detecting movement, light, proximity, location, etc.
- Vibration motor
- Logic board (the computer)
- Memory
- Battery
- SIM or other memory card (contains provider data and contacts for certain cellular types)

Device Optimization

Have you ever noticed that apps seem to perform more slowly when the device is hot or when there is a low battery? There's a reason for this: These are two conditions in which the device actually slows things down to help.

Hot Device

If a device gets hot, it can overheat and get damaged, so many devices have an optimization system that does things to help cool down the device. The device may slow the CPU, free up memory and turn off parts of the system to help keep it from overheating.

Low Battery

Similarly, when the battery is low, the device may turn off systems to try to conserve battery power. If your app depends on a system that helps it connect to a network or the Internet or that gives it faster performance, then that system might be something that drains the battery, so the device may try to reduce its use or turn it off.

Light

Different lighting conditions have an enormous impact on our ability to view what is displayed. Devices have a light sensor that helps determine how bright or dim the lighting is around the device, so it will automatically brighten or darken the screen depending on light conditions. If your app or web site is very light in color (eg. white background with light grey font), it might be visible under artificial lights indoors, and in dark conditions, but when the screen is dealing with bright sunlight outside, the combination of bright light and the light sensor may cause your app to be invisible under those conditions.

Proximity

The most expensive activity on a device is lighting up the screen. It will use various cues such as user inactivity, a locked screen, and whether you are holding and looking at the device to decide whether to turn the screen off to save

battery power. The proximity sensor can tell if you are near the device or not, so if it senses that you are not near it, it will signal the device to consider going into screen off optimization mode.

Mobile Apps

The apps we test are computer programs, which at their essence are a set of instructions written in a language that a computer can understand. The fancy, whizzy features that we enjoy on our devices are provided by the same people who supply us with different operating systems, like iOS from Apple, Android from Google, and many others. Programming frameworks provide app designers and developers with different languages and features to help them create a great mobile experience for us. What follows are some of the features that are available. It would be rare for one app to use all of these features, but it is possible. Usually, the app you are testing right now will utilize some of them, so it's important to work your way through this section of the book.

Don't worry if it doesn't make complete sense at first. You may find it helpful to reread or review it when you are faced with this technology on an app you are testing.

User Interface (UI)

This is what is what you look at and interact with. It's hard to miss—it's staring right at you!

UI Components

An app designer and developer has a lot of components to work with when designing an application. Grid systems for creating just the right layout and and design, custom images and media plugins for video and sound. Development frameworks provide different mobile-friendly components such as buttons, sliders, links, pickers, toolbars, radio buttons and others. You can also create different ways of messaging the end user using popups, banners or other ways of notifying them when something important occurs.

UI Characteristics for Apps

Mobile User Interfaces are a special challenge, especially when we are used to larger and larger flat screens for PCs. Mobile devices have less space for apps, and there are a lot of different screen sizes and screen resolutions to take into account. Some smartphone devices can be very small, with low resolution screens of 480x720, while others might be larger with far greater screen resolution. Tablets can have much larger screens, with the popular 7" tablets at the smaller end of the scale. Some phones can have quite large screens, and they are sometimes called "phablets" since they are in between a phone and a tablet. All modern tablets and smartphones tend to have a limited size compared to their PC cousins. Application focus is based on one window or screen at a time, so the app focus can only be on one thing at a time. PCs on the other hand, usually display multiple options at once in the same screen.

App designers have to cope with lots of screen sizes, lots of resolution sizes, and a more cumbersome workflow with one screen or window at a time.

What Can Go Wrong in the UI

Since workflows have to be broken down into something simple, and across several screens, it can be difficult to design mobile apps or web sites/apps without burdening the end user. There may be too many components or too many items displayed that are hard to see or interact with, especially while the user is moving. Many people like to use either portrait or landscape, and some people prefer one or the other. Apps may only work in one orientation, and not in the other, or they may be inconsistent, sometimes working in one orientation and not in the other. Not supporting both orientation types on every screen in an app will make it hard to use for people who prefer the one you don't support.

Sometimes designers target a larger screen in a design, and the app looks too busy, or elements are too small when displayed on a smaller screen. Conversely, if an app is designed for a smaller screen and not tested on a larger screen, elements may be inappropriately sized, or laid out awkwardly with a lot of unused space that results in an awkward impression.

Error handling and workflows can be improperly modelled, or not modelled at all in the UI. What happens if you

change your mind and want to go back at any point while using the app? Can you, or are you trapped?

High-resolution Touch Screen

Touch screens are used for both display and inputs on smartphones and tablets. This requires a combination of hardware and device features and an operating system as well as software to support. Within an app, we can enable or disable certain kinds of gesture inputs.

Touch Screen Components

The touch screen has very tough glass, sensors to determine input pressure, location, and movement on the glass, and a system underneath that interprets those inputs. Some sophisticated devices are able to discern between accidental movement and touching, but it's still a good idea to try out accidental gestures to see how your app or web site handles them.

The screen is supported by a system to support touch inputs and gestures. Several physical sensors are used to support this. Some touchscreens use technology that requires pressure from fingers or other input devices like a stylus, while others utilize pressure from touch as well as the electrical charge our bodies give off to detect input. When touch screens support electrical discharge, or capacitance, they tend to be more accurate and easier to use than when using pressure sensors alone.

When you move the device from one view, such as portrait (up and down) to landscape (device is on its side) you will notice that the screen will usually adjust and redraw to orient itself to the new view. Auto-rotation, auto-sizing of an app in a view is provided by the operating system, and requires a combination of movement sensors and stabilization features. Some app developers turn this feature off, but that means they will exclude a good number of people who prefer to use the device in one mode versus another. For example, landscape is often turned off, but many people use their devices in landscape mode.

Devices have a light sensor to brighten or darken the screen automatically depending on what the external lighting conditions are. If you are in the dark, it will brighten, and if you are in very bright light (such as outside in sunlight), the screen will dim to a higher contrast so that you can see it and interact with it.

Touch Screen Characteristics for Apps

Touch screens have a hard job, they need to display whatever is going on to the user, and they also need to capture various inputs at the same time. When developing an app, we need to realize that not only will people be looking at the app, they will be trying to input into it, and reading corresponding changes and outputs, all at the same time. That has an impact on display, especially since we have smaller screens than PCs, and a large range of screen sizes to support with smartphones and tablets. Can people

see the app components when the device is moving? Can they read and interpret information? Can they input and interact with it? Programmers can utilize a lot of different gestures within apps to support this, or provide very few.

What Can Go Wrong with Touch Screens

Touch screens are resource-intensive. The device has to work hard to light up the screen, draw the app for the user to see and interact with it, and it relies on sensors to capture inputs. Now imagine that you are moving, and the device has to maintain connectivity as you change networks, and movement sensors are coming into play, as well as proximity and light sensors. As you try to view and interact with the screen, a lot of activities are taking place on the device, and that can really stress it out, especially if our app is also resource intensive.

Here are some things to watch out for when testing:

- **App image resolution not suited for screen**: It was developed for a higher resolution than the current device, especially smartphones.
- **Touch/gesture inputs poorly laid out, hard to use**: Touch targets might be too small, so they are difficult to trigger, or they are too close together and too numerous, triggering unexpected events or side effects.
- **Rotation not supported, or causes problems**: Some actions are easier to perform if the device is on its side, and the app may not support a more usable

position. Rotation may also cause problems with inputs, redrawing the screen, or other physical/virtual interactions that occur. For example, submit a form to the web while rotating may tax the system and cause problems.

- **Input lag**: Too much processing or memory usage by the app itself can cause inputs to lag due to resource contention.
- **Device freezes up**: Errors during inputs, display or other features may cause performance bottlenecks cause the device to freeze up and become unresponsive.
- **Sore fingers from repetitive input**: Excessive need for typing and gesturing can physically hurt after usage. This usually points to an app that is too complex, or does not use mobile affordances enough to help the user reduce their input requirements.

Device Sensors

Sensors aren't much to look at if you take a device apart, and some of them are tiny, but they play an enormous role in our mobile experience. In fact, access to inexpensive sensors are what makes smartphones and tablets so compelling to use. Now that we can get cheap access to sensors, we can combine them with prior technology to create new, powerful user experiences. PCs do not have these types of sensors, so we didn't need to worry about them before. Mobile devices depend heavily on sensors, so we need to be

aware of how they work and incorporate that knowledge into our testing.

App designers and developers can tap into many of them to help create more interesting and more user-friendly mobile experiences. This access is provided by the development framework that the manufacturer of the operating system publishes and supports, called an application programming interface (API). Different devices have different combinations of sensors. We'll look at some of the popular ones.

Touch Sensors

Touch sensors are built in and are part of the physical device. They are part of the touchscreen input and sense either pressure or electrical capacitance when a person interacts with them. The resulting interaction is sent to a controller, which is then processed by the device. Developers usually have access through these indirectly, such as through gesture APIs, so they can create their own special inputs.

What Can Go Wrong with Touch Sensors

One of the biggest problems with touch screens are apps that are not designed to handle inadvertent movement. We use devices on the move. What happens if an app is activated while the device is jostling in a pocket, bag or purse? This is the dreaded "pocket dial" scenario. Some mobile phones were notorious for accidentally dialling

emergency services, such as 911 or 999, just because they were jostled or accidentally bumped. Apps shouldn't delete data or start up anything that could cause a user problems through unintentional touch sensor interaction.

Movement: Accelerometer and Gyroscope

These are sensors that detect movement and are used in applications. Most devices have accelerometers, while more high-end devices will also have a gyroscope.

- **Accelerometer**: A sensor that detects a change in movement. It's also used used to orient devices by measuring gravity.
- **Gyroscope**: A sensor that detects 360-degree motion. It measures rotation by factoring in things like yaw, pitch and roll. Gyros have been used heavily in aircraft for many years. It's amazing to be able to have them inside devices that we carry around with us.

How Accelerometers and Gyroscopes Are Used

App designers and developers might tie into these sensors for the following reasons:

- To determine app orientation (portrait or landscape display)

- For stabilization of image or video display
- To detect movement for app interaction (tilt, rotate, shake, etc.)—for example, a tilt is often used in games as an input, and a shake is often used to clear a text field or multiple text fields in a form
- To help determine location and pinpoint direction of movement, where the device is pointed, etc. to help provide more accurate information.

What Can Go Wrong with Accelerometers and Gyroscopes

When you are using movement sensors, the device is working harder than when it is sitting, stable and unmoving. Orientation problems may leave the app upside down or locked in transition, or it may not display properly if the movement is triggered while an app is working hard, or using up too many device resources while the sensors are also engaged.

Rapid switching from portrait to landscape with slight or no movement can cause an app to freeze up or go into an error state. This can occur when you are holding the device at an angle, especially when lying down, or lounging on a surface rather than being upright.

Since the device is working hard, if you are doing other activities at the same time, the app may freeze if too many activities are being processed at once. For example, if you are moving the device while gesturing, and it changes

orientation, the app might overload and freeze up. This is common when there is movement that causes inadvertent inputs: while walking, in a vehicle that is moving up and down, or simply when handing the device off to another person.

Movement: Magnetometer

This is a sensor that measures the strength of magnetic fields. As you may remember from grade school science, this can be used to determine direction.

How the Magnetometer Is Used Apps that require compass information for navigation will use this sensor. It can help determine what direction your device is facing: north, south, east, west, and combinations of those four.

What Can Go Wrong with a Magnetometer

A magnetometer can get interference from steel or other magnets just like an analog compass, or it can have trouble finding a point due to too much movement. Have you ever walked out of a hotel in an unfamiliar city and been unsure of your orientation? If you are around structures with a lot of steel, your magnetometer may have too much interference to work and you may have to ask for help.

Movement: Touchless Input Sensors

Touchless gestures are starting to become popular because they are convenient, and can save our finger tips from strain. We can gesture, or wave our hands *near* the device, and interact with it without touching it. Other body movements can also be captured as gestures such as head, or even eye movement. To capture and support this motion, devices may have specialized sensors, or may simply combine some of the sensors that we have mentioned above: proximity, light, and even capacitance and the camera might be utilized as well.

How Touchless Input Sensors Are Used

Apps can be interacted with by moving your hands (or other appendages) near the device, rather than touching the device. This is incredibly powerful from a usability standpoint. It can be very convenient to just wave your hand near the device to start a program, or to change app conditions or behavior, before you start touching it. It can be a time saver, especially if you are doing something else while using an app, and you aren't able to hold the device and focus on it completely.

What Can Go Wrong with Touchless Input Sensors

Motion sensors are fairly complex, and are susceptible to interference (other things moving, electrical interference,

etc.) in the environment where they are being used. Sometimes different lighting conditions can cause them to malfunction. Electrical interference, or a lot of movement can cause problems as well, since the device is processing a lot of things while it is moving, so there might be performance issues or lags during input, or the device may mis-interpret touchless gestures.

Using Speakers and Mics: Voice and Air Pressure Inputs

Mobile devices have speakers and microphones to capture and output sound. Sound waves also cause pressure, so these devices are sensitive and utilize movement in their design.

How Speakers and Mics Are Used

At their simplest, microphones and speakers are used to detect and send on sound signals, or to play them back so the user can hear sound. Apps may utilize voice control (another touchless input type) to operate, or change behavior, relying on keywords or phrases. In fact, many modern devices have voice control built-in, so it's a good idea to test that your app or web site can be accessed with built-in voice control.

More complex uses are to use speakers and microphones (mics) to capture air pressure as inputs. A user might interact with your app by blowing into a microphone or

a speaker. Since sound inputs and outputs are very similar, both can be used as input devices. The air pressure from the breath of the user can be interpreted and used as an input into an app. So far, I've only seen them used in gaming applications, but there are a lot of other potential uses.

What Can Go Wrong with Speakers and Mics as Inputs

Apps may simply not work with built-in voice control because no one thought of that as a way to launch or interact with an app or web site.

If they are used within an app, they may have problems with interference from other devices, electrical interference, or distortion. There might be performance issues while capturing or playing back sounds, or there might be delays.

When used to capture breath as an input sensor, there might be overload from inputs (especially while moving), or performance issues. Since breathing is a normal thing to do, we also might accidentally enter in an input into the device if we are holding it closely to our face, causing unintended results.

Communications

Smartphones and tablets are communication devices. In fact, smartphones are a technology mashup of two devices,

a computer and a mobile phone. (Well, there's a camera in there, too, and some other things, but what makes it a "smartphone" is having a computer and mobile telephone in one package.)

Mobile phones have been around for a long time. They date back to the 1940s, when they were developed for military communications during World War II. The first commercially available mobile phones became available in the late 1970s. They use radio waves for receiving and transmitting information. In fact, people in the wireless industry refer to mobile phones as "radios" because that is essentially what they are. While they look different from the box on Grandpa's kitchen table that booms out talk radio each morning over breakfast, they use the same kind of technology. Imagine being able to not only receive messages with that kitchen table radio but also transmit them, and you have a rudimentary mobile phone. In fact, a mobile phone is a sort of technology mashup as well: a telephone and a radio.

It was mind blowing in the early 1990s when mobile phones became portable and reasonably affordable. One of my friends was the only kid in our school with a mobile phone, and he hooked it on his belt with great pride. It was also about a foot long, not including the antenna. Prior to that design from Motorola, mobile phones came in a briefcase and were used by wealthy businessmen when they were on the road. As more features were added and technology improved the devices (smaller size, better battery life, more

cellular networks), we saw the inevitable combination of a small computer and a cellular phone. Smartphones didn't really take off in the market until Apple released the iPhone in 2007. Now they are everywhere.

Tablets aren't new, either, but the ones that we use now are essentially larger smartphones without the phone. In the past, they were very expensive, portable PCs with touch screens. But, once again, Apple disrupted and changed things. Now, the most popular tablets use operating systems that are based on the smartphone technology, not the PC technology.

I've discussed the history of smartphones because it is important to understand how foundational communication components are to mobile devices. Today, we have branched out to text messages that use the same technology as cellular voice traffic—usually short messaging service (SMS) or multimedia messaging service (MMS)—and technology that depends on the Internet: instant messaging, voice over IP (VOIP) programs, video chat, and email. There are also application communication services that alert you of information you might be interested in, such as weather alerts, stock information or sports scores.

Communication technology remains a foundational aspect of mobile devices, and new technology using the Internet is being created to enhance and expand our ability to keep in touch or be informed. App designers and developers integrate with these services when they create apps to harness this popularity and functionality.

Telephone

Smartphones come with a telephone application built in. This requires hardware and software, such as: microphone, speaker, antennas and wireless technology to support voice communications. Applications to manage phone calls, filter noise, optimize communication networks, etc.

How Phones Are Used

App designers and developers can integrate mobile phone features into their apps to make a call from an application. They may even integrate with contacts stored on the device to call someone to share information, solve a problem, etc. They may use it for customers to engage with the company, or to collaborate with others within the context of the application.

Note: Some apps may use alternative communication protocols such as Voice Over IP (VOIP) to provide free voice communication over the internet, bypassing the telecommunications network, company and associated costs. They still use the sound inputs and outputs of the device to support this though.

What Can Go Wrong With Phone Integration

The app calls a wrong number, or it uses phone formats for a specific locale, so it doesn't work in other locations. For

example when you are traveling or live outside of North America forcing customer service contact through a 1-800 number in an app won't work. 1-800 numbers only work in North America.

Phones on devices are a low-level, foundational application (the phone part of smartphone), and on some handsets they operate outside of normal operating system that we are used to interacting with during application operation, so they may require special effort to interact with smoothly. Interruptions from phone calls may not be handled well by the app you are using, and it may crash or freeze up.

Text Messages

Smartphones and all mobile phones support SMS (short message service) or MMS (multimedia messaging service) messaging. App designers and developers utilize it because it is supported on every mobile phone and has an amazing interaction rate.

How Text Messages Are Used

Text messages are powerful engagement methods, so app developers are finding that they can use them to remind people to use our app if they haven't used them for a while, or as an alternative method for providing information to end users. They may use them as a form of reminder or even advertising of other products and services that our

organization provides. They can also be used as notifications of important information, such as current events, or changes in the system that we should be aware of.

Sometimes they are used to supplement authentication or logins. Administrative tools may send codes to allow access to systems or services using text messages, particularly prefixes for passwords.

Text communication with others is incredibly convenient, so it is a natural fit within apps. If you want to share information quickly, text messages are one option. You can incorporate media attachments like images and video files to enrich communication. Imagine that you see something important, and the app allows you to take a picture with your camera and message it quickly to someone who needs to see the information. It's a powerful, fast mechanism for sharing information.

What Can Go Wrong with Text Messaging

Text messaging depends on integrating with built-in technology on the device, and the reliability of your cellular or network connection. There are a lot of factors that can impact the effectiveness of using texts within an app.

If your app or system depends on messages arriving quickly, you may run into problems with time delays. There can be a 72-hour delay before you are notified the message delivery failed. Some providers provide a two day window to send and receive a message.

Users can turn off messaging or change permissions on the devices themselves, which can cause messages to be rejected, or messages may not get through. If the app depends on the service to be running, it may fail in unexpected ways if the services are turned off.

Messages may come through garbled or incomplete, which is especially problematic when there is a lack of localization support for languages with special characters outside of English characters.

Large files sent as text attachments can cause the app to perform poorly, freeze up, or use up a cellular data plan too quickly.

Another side effect of messages, whether they are incorporated in your app or not, is that the interruption by a text message may cause your app to malfunction. It might freeze, log out a user from a server, or not be able to redraw the screen after the interruption.

Instant Messaging

I am making a distinction between text messaging using cellular technologies, and instant messaging using networking technologies. There are a lot of apps on our PCs that support instant messaging or chatting using a network or Internet connection. Many apps on mobile devices support this as well, including within apps whose primary purpose isn't messaging.

How Text Messaging Is Used

Apps can send text messages over the web (and some apps support files including images and media as well) between people or groups of people. This provides a simple, quick communication interface for people, or limited groups of people.

What Can Go Wrong

Text messages depend on networks to work properly, so if a network is slow, has errors, or if an end user is switching networks while moving, text messages may not arrive when you expect them to due to time delays. At worst, messages aren't delivered at all, or fail part way through transmission. Attached files in a message might not come through, or are corrupted in transfer.

Also watch for message notifications aren't recognized on the device when the application is currently displayed on the screen. Even more simply, yet no less annoying, messages might be too intrusive or interrupt other activities such as phone calls or other activities on the device.

One problem that custom messaging can cause is due to the interruptions that messages can cause while using the app. Sometimes the app can't transition away from the message back to what you were doing before without causing errors.

Video Messaging

Many devices now have front-facing cameras that allow you to video chat with other people.

How Video Messaging Is Used

The combination of front-facing cameras, and systems to support video capture and transmission allow you to see the person on the other end of the line when you communicate. Any collaboration activity or application that uses voice or text communication can be greatly enhanced by using video. Communication apps often support this feature, but productivity and medical collaboration apps are now appearing as well.

What Can Go Wrong With Video

Video is very data intensive, and requires a good, stable, fast, reasonably error free network connection. If there are network problems, video will have a lot of issues, such as distortion in voice or video, or both, or even dropped connections. Video is also resource-intensive, so it can suffer from poor performance, lag or delays in voice or video transmission.

If the app is already resource intensive, the extra resource usage from video may cause it to freeze up.

If the app uses video over cellular networks, it could rapidly using up cellular data plans with the large amount of data

required for streaming. It can also wear down the battery quickly (using display, camera, video streaming, heavy network use all at once.)

Media

One of the enormous benefits of mobile devices are their support of rich media— audio, video, games and animation and high-resolution images.

Audio

Smartphones require audio equipment so that we can talk and listen to others on cellular phones. We can also use that technology for other purposes on mobile devices. Devices have built-in microphone for inputting or recording sounds, speakers for listening in real-time, or playing back recordings. There are also input jacks to support input and output accessories such as specialized microphones and speakers that can be plugged into the device.

The operating system provides APIs for managing sound files and there are also specialized applications for storing, organizing and playing recordings like music and e-books. There are also APIs available for modifying or adding sound effects.

How Audio is Used

Apps can play sounds to alert users or for actions completed in an app as a way of providing rich feedback.

Apps can allow you to record, store audio files— music, sounds, voice dictation, messages, etc. for later usage, or to share with others.

There are also apps that support audio playback for supported file types for music, readings, voice recordings, etc.

What Can Go Wrong with Audio

There can be a lag in playback or recording, or even worse, distortion in playback or during recording.

Audio can be resource-intensive, so watch for poor performance or device freeze ups. It is easy to create large files, so the app or system may have difficulty saving them. THat might lead to corrupt or deleted files when saving. Not respecting operating system file types and storage rules can also be problematic.

Digital Rights Management (DRM) and similar licensing restrictions prevent media access in certain countries or regions may cause problems. Audio may not be available for everyone who uses the app, or the app may not handle a restriction gracefully when it expects the content to be there.

Video

Video is supported by a combination of technology - camera(s) for video, microphone and speakers for sound, and if streaming, network connectivity.

There are also applications for storing, organizing and playing video recordings and built-in file system and storage memory provided by the device.

How Video is Used

Video is a rich medium for sharing information, or communicating. Apps can record, store, share or stream video. They can be integrated into an app for entertainment, sharing information, interviews, demos, how-to guides or anything that needs to be shared with others. Apps can record and then send out video to others via text, email, or within an application so that something important can be captured and shared. Other apps may also use video streaming for face/voice communication within the context of app usage.

What Can Go Wrong with Video

Much like audio, video is a resource intensive activity on a device, using up memory, processing, rendering activities, storage considerations, and when sharing or streaming, it is very bandwidth intensive on a network. This can result in poor app performance, lag in recording or playback or freeze ups on the device. There might be problems rendering or displaying the video, so the screen is garbled or blank on playback.

Videos may have distortion (visual, sound or both) when recording so your video file or streaming experience is unusable.

Media files created by video recording might be too large, causing errors when trying to save or utilize them.

Sound interference when recording such as an annoying beep or buzz at an inopportune time might be difficult for apps to handle. There might be interruptions that cause freeze ups in the app, failed processing or saving, or distortion in a file or screen.

Different devices and programs record and process video at different rates and resolutions, so something recorded on one device type may not work on others. Also, an app may not respect the restrictions of the operating system on file size and type when trying to save them, which can cause errors.

As with sound, Digital Rights Management (DRM) and similar licensing restrictions may prevent media access in certain countries or regions, which can cause problems with usability, or poor error handling if the app is expecting the content to be there.

Animation and Gaming

Smartphone development platforms provide libraries for animation and gaming as full APIs. They are rich applications with a lot of color and high resolution graphics, animation, as well as things like sound and video. They tend to make full use of inputs like gestures and other affordances that make interaction simple and fun.

Gaming technology isn't just used for fun though. Animation and gaming APIs are now used in visualization fields, such as rendering for architecture and design of buildings and systems, medical imaging, product design and aviation.

How Animation and Gaming Tech Are Used

When things move on your screen in an app, and you aren't playing a video, animation technology is being used. Things can move, sounds can play, and your experience on the device is richer, if it is done well. Gaming technology also supports inputs and interaction by the user, so you can touch, tap and move the device around to interact with it and change what is occurring on the screen. Gaming libraries that programmers can use tap into a great deal of mobile technology, making great use high resolution touch screens, movement sensors, and even things like location services and communications. Many games are pushing the envelope of user input - I've seen games that make use of the accelerometer and gyroscope to support moving the phone to interact with an app, and even measuring pressure from a user blowing into a microphone as an input type.

It isn't all fun and games though, serious applications like medical imaging and aviation mapping and others use the same technology that games do. Since they are serious apps that require a lot of information and accuracy, they rely on rich media experience and interactivity. They may also use

gaming or animation libraries for 2D and 3D rendering of images for visualization fields.

What Can Go Wrong with Animation and Gaming Tech

Animation and gaming technology can be very resource heavy. That means the device has to work hard to create animation, utilize sound, and capture various user inputs. THat means that the CPU, GPU, memory, and everything else is working hard. If networking is also needed, then the device is going to be stressed to its limits at times. Watch for poor performance, apps freezing up, and lag on inputs or in animation. Also watch for problems rendering animation or transforming images to 2D or 3D on devices with less processing power or memory capabilities, smaller screen sizes and with lower resolution than others. Sometimes it's as simple as images or animation created for larger screen sizes. There can be problems due to size with smaller screens, especially when required gesture movements are too large, causing fingers to go off screen. Touch targets might be too small, creating difficult user interaction.

There may also be rendering problems, such as distorted pictures on the screen, or strange colors like grey, pink or green if there are video rendering problems. Also watch for localization problems - your app may handle characters in different languages as text, but the rendering engine may not handle them when they are part of images.

Watch for problems with combined activities - lots of ren-

dering, animation on the device, as well as user movement and interaction. These sorts of scenarios can cause issues, particularly if the device also has to adjust to orientation changes.

Cameras

Cameras have become a staple component in mobile devices. There is a combination of hardware (a powerful camera), a program to manage it and operating storage for photos or media. Your device may also sync up with servers in the cloud, or other storage via a network. Typical components include:

- Camera hardware, lens and flash (usually an LED light)
- Image sensor (the digital equivalent of paper film)
- Systems for autofocus and image stabilization
- Programs to support:
  - Photo capture, editing and storage
  - Video capture, editing and storage

How Cameras Are Used

Cameras are used in a variety of ways within applications. They can be incredibly useful to capture information to share or send to others, or to integrate within an app for live information, or just to create a document. Within an app, you can use the camera system to take and edit photos,

record and edit videos, save them, sync with other devices and systems, or simply use them in the context of an app. They can also integrate with communication services to chat using live video for face-to-face communication, or to send video files to another person or system. The camera can also be used as a scanner, whether to capture information using shapes or OCR (optical character recognition), or to scan barcodes or similar technology such as Quick Response or QR Codes to provide you with information or other resources.

Since the camera is made up of several components, app developers have figured out how to use the technology in some surprising ways. Some apps may use the LED light in the flash as a light source such as a flashlight. Others use the camera to help people measure distances or angles. Some augmented reality apps allow you to scan what is currently around you, and then enhance and add in features or objects. This is useful for people who are planning on renovations for their home, or for augmented reality game (ARG) designers. Cameras in smartphones can even be used to measure pulse and heart rate. Turning on the video camera with a front-facing camera can act as a handy mirror if you are on the move and worried about your appearance.

What Can Go Wrong with Camera Integration

Cameras can be awkward to integrate with. They have their own hardware and system to manage them, and interacting with them through an app can be problematic. If your app takes you to the camera function, and does not provide any navigation, it can be confusing to users. If you are interrupted, reload the app and it is in a camera view without any navigation, you may forget what you were doing with the app, or get confused. Cameras can take a lot of resources, particularly when capturing high resolution images, or when recording video. That can result in poor performance, and freeze ups when capturing or saving images. There might be image capturing problems with your app: the camera seems to work fine, but the images are blurry, distorted, or difficult to capture without delays and lag. Video can also suffer from this problem, especially if your app is already resource-heavy when the camera is in operation, which can result in freeze ups or distortion, or the video may appear to jump around or skip.

There can be hardware integration issues as well. Some devices have both front and rear-facing cameras. An app may activate the wrong one or both when it shouldn't. If sensors are being used for something other than image capture, there may be a malfunction or strange behavior, particularly after operating system upgrades, or on hardware that is different from what was developed on.

Location Services

One of the most convenient and powerful features of mobile devices is their ability to pinpoint our exact location at any give time. This is useful to help us find our way if we aren't sure where we are going, to find local services or get up-to-date information on things like traffic, weather or local specials on products, services or entertainment.

Location Components

Location can be determined in several ways. Some systems use only one technology, but many apps utilize several at any given time. GPS (Global Positioning Satellite) is the one most people think of and assume is being used to determine location, but it is just one of several. GPS on your device works by having its own dedicated technology. First of all, there are satellite navigation systems that use near-earth satellites in orbit around the earth to help track locations. Your mobile device has a GPS receiver and wireless system to support the technology, as well as an antenna and specialized processing. GPS depends on at least one satellite to help get a bearing on your location, and the more satellites used, the higher the accuracy. The system can "triangulate"", or calculate distance based on the time it takes for radio signals to be processed. Time is another important component in GPS to help accurately pinpoint your current location.

GPS can be expensive (it takes a lot of processing power

and time to contact near-earth satellites, and calculate and process results), so we can use similar technology based on cellular towers and your current location. An app can determine your current location based on the cell tower you are currently connected to. This is not very accurate though, so triangulation can be used, based on multiple towers nearby to calculate more accurately. In this case,

Another wireless technology that can be used is the physical location of the current WiFi device you are connected to.

Some services will combine the above technologies above to get better accuracy or faster service. For example, "assisted GPS" may use WiFi locations or cellular location services in addition to GPS. For even finer details, such as what direction you are currently pointing the device, movement sensors like the accelerometer, gyroscope and magnetometer can also be used. This can be incredibly useful if you want to get very accurate location information, particularly if you are using geofencing. With geofencing, if a device enters a certain region, an app can change behavior, or provide different information.

Some non-wireless solutions include using a device ID stored in your mobile device based on the address where you registered the device, and your web Internet protocol (IP) address. This is assigned by an Internet service provider when you connect to the web and can be looked up in various systems that record these addresses.

How Location Services Are Used

Location services are used in a variety of ways. The simplest can be to provide your current location to an app without forcing the user to determine it and enter it in themselves. This might be in a form to submit to a company or service, or to alert the application to change the information it is providing to you.

Location services provide help with map apps and to provide directions to map out trips for users. This is one of the most common uses. However, they can be used to drive custom content for services or products nearby, or to change the information that you receive so that it is more relevant, especially when you are on the move.

Location services can be used to promote products and services. If people are near or inside retail stores or restaurants, they can use that information about users to help encourage people to visit, or to promote people to purchase of certain items. Geofencing is a technology that can provide custom content based on location, so if you are a shopper in a store and they have an app that uses geofencing, they can point you towards things you might be interested as you move around the store, in real-time. Geofencing is also used for other purposes, I have seen it in environment, health and safety applications and others. As a user moves, the location and context around them has a big impact on what information is needed, or how an application behaves. It can be an enormous time saver, or convenience for people to have things update based on where they are moving.

Games, entertainment, media and streaming often have location-based promotions and, in some cases, restrict certain content based on your location. You can play games with people around you, or have the content vary depending on where you are. Even simple information such as what languages or locale for the game to operate in can be utilized as well.

Other apps, especially social apps, use location services to share and view information from people around you, like statuses, images, and other things that we might be interested in. Many social apps allow you to "check-in" or notify others when you arrive at a particular location, and this is often used by businesses to help promote themselves on social media.

What Can Go Wrong with Location Services

This is an area that can have frustrating or hilarious results if the systems get it wrong. One of the apps we tested during my Mobile Applications Testing course in 2011 was a public transportation map that used location services. It would find your current location and then point you to the closest public transportation options. Sounds great for travellers, right? Trouble was, it seemed to be accurate to about two to three blocks. That's fine for some applications, but it's completely unacceptable for a service that is meant to help me find the right train or bus. An incorrect location such as a wrong address, or I am not where the app thinks

I am, can be incredibly problematic for location features in an app. Some apps are ok with an address that is close enough, while others will fail catastrophically if it is even a little bit wrong.

Some devices attempt to guess your most recently used WiFi locations if network or location services aren't available. This does not work very well when you are traveling and you haven't had successful network connectivity.

Performance can be a problem because it is a lot of work to determine location both on the device, and while utilizing other services. GPS is inherently slower because of physical limitations. For example wireless signal speed with GPS takes longer because it has a longer distance to travels through the atmosphere to satellites. It is also susceptible to interference from cloud cover, tall buildings, and poor weather. Any location services that use wireless are also susceptible to interference in connectivity. Steel, electrical activity (either in the atmosphere, or due to other devices around you) and even low geographic areas can cause problems with wireless signals that location services depend on.

Movement can have an effect, especially if you are using location services that calculate your current location based on getting information from multiple sources. As you move, expect errors when stopping and starting a lot in a vehicle or while walking. You may inadvertently end up in an area that has interference and the location services loses you momentarily. Some location services may have algorithms that take your movement into account and an-

ticipates where you will be next, and miscalculates your position when you stop or change directions frequently. This often appears as your current location jumping around. It may move around wildly as the device struggles to calculate.

As you use more technology to help determine technology, the app or even the device itself may freeze up when trying to obtain location if there are problems. This is due to a lot of technology being used at once, and poor error handling when any of the mix of tech that an app depends on fails.

Map Integration

In many cases, the real power of location services lies in their integration with mapping software. Smartphones (and some tablets) provide a default map application, and the development platforms provide APIs so that designers and developers can integrate maps with an app. It can be an incredibly useful feature to help guide users to a local service or your office or other services. They can be fantastic navigation aids.

Mapping software depends on data and imagery that can be gathered in various ways. The technology that underpins mapping software is GIS (Geographic information systems.) There are satellite images of the earth that can be translated and annotated with mapping information including landmarks, municipalities, roadways and other man-made or other structures and features. Data is also captured by airplanes that take pictures of the geography,

by vehicles that drive around and capture data by taking photos and latitude and longitude readings, and some are even crowd-sourced, relying on user-submitted information.

How Maps Are Used

Maps are usually used in conjunction with location services, but they are sometimes just used as a reference without taking your current location into account. Within an app, it is common to launch a map view to help users find things that they require that are nearby, or to simply plan a trip of some sort. users can interact with maps using map-supplied gestures to zoom in or out, or to look at other areas of the map, and app developers can also add their own custom gestures to add more power and control if required. Another area app developers can add more information within the context of an app and the user's current needs are by using annotations. You can use the built-in annotations that are part of the map, or create custom annotations to add more information about locations, organizations, products, services etc. in the user's current current view

Maps are a powerful tool, and it is amazing that we have so many options at our fingertips within these devices.

What Can Go Wrong with Maps

Maps are complicated to do well, and when integrated with software, can have problems. Mapping data might be out

of date, with information that has changed or is no longer relevant. Many of the software systems we use that supply maps free of charge do not use more expensive (and more reliable) sources of information, so there may be areas of inaccuracy. There is nothing worse than using location services that plots you on a map in the wrong location because of a problem with map data.

There can be a lot of map data to process, and your device will often need to download this over a network, so that can impact performance. Imagine you are on the move, and location services are trying to calculate your location, and map data is loading at the same time. This can lead to slow or sluggish performance, or at worst, freeze ups.

Your map may determine your home as a default location, which can result in bizarre problems when you are on the move if it doesn't determine your current location. Once, while travelling in London, UK, which is over 7000 km from my home with an ocean between us, my mapping software failed me. I was walking, in the rain, and I was lost and trying to find my hotel, so I used a map app to plot directions from my current location to the hotel. After a while, the map app just froze up. Once I finally made it to my hotel (after bribing a cab driver with an extra ten pounds) I connected to their wifi and checked my map app. A few minutes later, it displayed walking directions from my home in Calgary to my hotel in King's Cross London. It even showed directions of me walking across the Atlantic Ocean. Obviously, it couldn't map out my exact location,

so it defaulted to my home location instead of using better designed error handling.

There can also be simpler problems, such as integration problems if you use a map in your app. The map may not load properly (e.g., zoomed way out or in the wrong location), requiring user interactions to correct it. One of my friends developed an app to help you find services in a city while travelling, but it would load with a view of North America, instead of the current city you were in. It could take dozens of gestures to zoom in to a level that was actually usable. Also, transitioning to a map with no way of getting back to your app can be confusing for users. If you keep the map view within your app, people know the context of usage and what to do. If you leave your app and launch in a map, people may get confused, especially if they are interrupted and return to your app and it just keeps launching in a map view with no way to navigate back to what you were previously doing in the app.

Networking and Internet Access

It's hard to imagine now, but we used to use computers that weren't connected to other computers. The wide availability of the Internet makes this seem quaint. Many mobile apps depend on getting information from the Internet, whether it is from a server operated by the same company who built the app, from partners or from publicly available sources. Apps can get updates to information or even to the apps themselves in various ways. In fact, when you start an

app, it might immediately connect to the Internet, talk to a server and update its information, all before you start using it.

Some apps are just an empty shell without an Internet connection. Each action or page that loads is filled with content from a source outside the device. Whenever you finish typing on your device and tap the "Go," "Return" or "Search" button, watch to see if it connects to the web. Want to find out for sure? Put your device in airplane mode and try using the app. If it works the same as always, then it probably doesn't rely on a network or Internet connection. Make a note if it fails at any point, because that's an area of the app you will want to test under different network conditions.

Networking Components

Networking requires communication systems from one device to another. Smartphones and tablets support internet protocols (languages that allow for the transmission of data over the internet) and networking protocols (languages for computers to communicate with each other.) Common internet protocols are TCP/IP (transmission control protocol/internet protocol) and the protocol that the web runs on is HTTP/HTTPS (hyper text transfer protocol.) The devices themselves depend on wireless communication hardware and systems and services to help determine optimal connections to use. Without ideal wireless conditions, networking will not work properly.

How Networking Is Used

Any time we need to send or receive data outside of the device, networking is used. Client/server applications are completely dependent on round trip communication because the app depends on a server to provide content. "Thin client" apps are another type of app that are completely dependent on getting content and information from a server to operate. They are "thin" because they just have a bare framework with no content. They work really well for maintenance since you only need to update information on a server and not update a lot of devices, but they require a network connection to work.

Any time you need to access or display information from the world wide web or the internet, whether you are surfing the web, or using protocols to connect and get information for an app, networking is used.

Some apps communicate with others who have the same app installed. Inter-application communication is one option app developers have to allow people to share information use or use in-app chatting or messaging to others in real-time. People also share information with those nearby using location services. Social interactions are powerful, we can check in at a location, take a picture and share with our friends, post messages and carry on virtual conversations with others.

What Can Go Wrong with Networking

Back when I started programming web applications in the mid-late 1990s, we were taught about distributed programming. Any time you need to connect to other machines and exchange information, it is important to put in proper error handling. Networks can have various speeds from blindingly fast to very slow, and they can suffer from latency problems, which is the time it takes for messages to be sent between machines. They can also suffer from "lossyness" which means parts of messages are lost and need to be resent. If networks are slow, if it takes a long time for messages to be sent on a network, or parts of messages are lost, it is incredibly important to have good error handling in your app. Now, on the web, we are so used to fast, reliable network connections, so people assume that it isn't something to worry about. However, modern wireless systems remind me a lot of the wired networking systems I worked with 15-20 years ago. They can be slow, unreliable, and difficult to rely on when people are on the move, or when networks get saturated from a lot of use. We've been spoiled when we create apps for the web; it will likely take a few years before wireless networks are as fast or reliable.

It is also difficult to transition from one sort of wireless system to another, for example walking away from a wifi source and switching to different cellular technology, in and out of dead spots, and back into wifi. (Imagine walking from your home to a local coffee shop that provides free

wifi, and changing from 4G LTE to 3G to a deadspot, and back to 3G on your way.) This type of situation can be very difficult for networking systems to handle. When you are on the move, so it can slow down the app. Programmers may rely on "caching," which involves temporary copies of data that can be saved and used on the device, saving network usage and speeding up performance. If the cached versions are outdated or missing, then an application can end up in strange states.

Apps may suffer from poor performance, time lags and delays if networking conditions aren't ideal. If apps require lots of downloads, or large file downloads, that can make performance much worse. If an app requires a lot of connections to a server, it might suffer if the device gets overwhelmed with information. Also, if the app is used on cellular networks, large amounts of data can cause user's cellular data plans to get used up too quickly.

If secure networks aren't used, private data exposed due to poor security in the app design. If private information isn't encrypted or protected properly, it can be easily intercepted using public networking infrastructure such as public wifi hotspots.

Notifications

Notifications are another convenience that let us know what is going on with a program or important events or keep us up to date with information when we're on the move. If an event occurs that we think a user would be

interested in, app developers can use this special kind of communication to interrupt them briefly and provide a short message. There are different options on operating systems that the user can choose for the format of the notification messages, with some of them interrupting more than others.

Notifications Components

To use notifications, programmers can create messages in a limited format, which can be displayed as a small alert message. Most Operating Systems allow users to choose how they see these messages. A data source is also needed, whether it is on the device itself, or comes from a server or other service over a wireless network.

An app may keep track of what it is doing, especially if it is running in the background, and alert you if there are any problems or when it has completed an important task. It may also store information locally in a database, and poll that from time to time, or get information from the Operating System.

For external sources, there is an extra dependency: the wireless network. Wireless conditions have to be reasonably good for notifications from external data sources to work. These sources might be web services or other technology purpose-built to easily share information about different events, or it might be as simple as making a query to a database system.

How Notifications Are Used

Notifications are quite simple: developers have code in an app that determines an event, either something that occurs locally on the device itself, or from another system that is accessed via wireless communications. For example, an app might track the stats of your favorite sports team, and notify you when they score in a game, even if the app isn't currently in use. A popup window or banner will appear with the information. Or, you might have home automation equipment that provides information about status or alarms to the internet of things (IoT), and your app might poll that information periodically, alerting you to any changes in home temperature, or whether a security alarm has been triggered. There are a lot of event-based situations that apps can inform us about: reminders, calendar events, stock prices, weather events, etc.

On the device itself, a program may work in the background, and alert you when it has completed a task or a calculation, or may monitor device conditions and let you know if you are using too much memory, or have too many apps open to function properly. Many error messages are generated as notification messages as well, especially if something went wrong with the application. They may also ask for permission to perform a certain task.

Communication apps and systems often use notifications, particularly if they are running in the background. Communication alerts can be quite simple: new email, voicemail, SMS or other messaging technology, or more com-

plex, which might be triggered by proximity or peer-to-peer events when someone is close by.

The Operating System itself will also provide notifications of different kinds of events or errors. Some of these include: low battery warnings, requests for permission, error messages, moving to a new network, losing network connectivity, and others.

What Can Go Wrong with Notifications

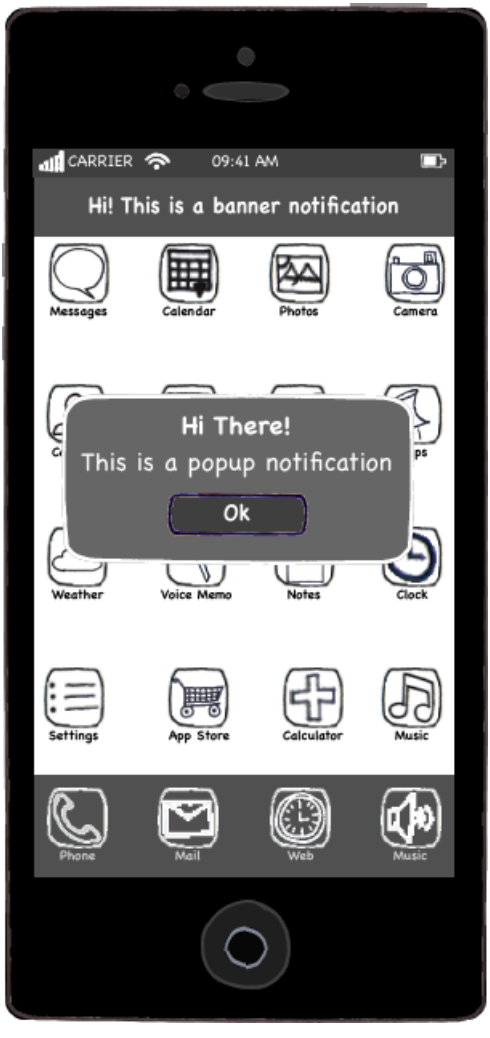
Notifications are an interruption to the current flow of what is occurring on the device. Larger messages like popup windows can take precedence and stop the current activities. This can be difficult for apps to handle well. The interruption might be unexpected at that point in app processing, and it may not recover. The app could crash, freeze up, or not be able to redraw the screen properly, forcing the user to restart it.

Integration with communications tech may fail, such as SMS, instant messages, call or voicemail notifications, email, etc.

If there are network issues due to wireless conditions not being favorable for connectivity and data transfer, notifications may not arrive when they are supposed to, or may not arrive at all. If a device loses connection, and reconnects, notifications may stack up and arrive too late, or all at once. There also might be unintended consequences under certain conditions when a user receives far too many notifications and it is overwhelming or annoying.

Data issues can occur if the data is out of date, wrong, or if privacy settings are bypassed or accidentally ignored. Utilizing new technology can sometimes cause surprises, if we have secured data using a mechanism that works for older technology, it might not work with new technology.

Remember to all the notification options your device uses. The image below shows two of them:



Banner and Popup Notifications Example

Data Persistence

When your app gets turned off, everything that went on in it will be lost unless the app saves some of that data somewhere to recall the next time it is opened. This storing is called "data persistence," and the data can be stored in different ways.

It can be stored as files in a file-system, or in a database that is either on the device, or on a server somewhere else.

Data Components

If you want to store data, you can save files on the device itself that the program can use. When data is saved on a device, there are usually strict rules about how large the files can be. This can take a bit of research to find out. When testing, the settings program on some devices will show you how much data your app is using.

Databases are specialized programs that store data, and relational databases have a lot of powerful features for storing and accessing data. Small databases can be installed with your program so it can access and use it. Other persistence models don't use the SQL language that relational databases depend on, and are quite flexible. They are popular with systems that deal with a lot of transactions such as social apps. Currently, they are more common on backend servers than on devices themselves, but you may also run into them on a device.

How Data Persistence Is Used

Any time we want to store data or information from one use of the program to another, or to share with other systems or people, some form of data persistence can be used. Here are some examples of data persistence that can be stored as files locally on the device, or on another system, within a database on the device or on another system, or a different storage technology:

- On device:
  - User data, such as user settings, user names and passwords preferences and colors, frequently used information
  - Saving user-created files, such as documents and media, notes, to-do lists
- Off device:
  - User data, files and media
  - User profile information such as demographics, commonly used data, loyalty programs
  - Outputs from productivity work
  - Purchase information of products and services

These examples barely scratch the surface of what our mobile systems can store.

What Can Go Wrong with Data Persistence

Data persistence can be tricky on mobile devices: there is less storage space available than on PCs, less processing

power and a lot of other technology coming into play while using data, restrictions on where and what you can store on some devices, and new technology quirks. When you need to save data on a different device or machine, networking comes into play as well. What happens to the data when the network conditions are less than ideal, or the device gets out of sync with the data source due to moving in and out of deadspots or weak network connections? Watch for poor performance, especially with a network connection, or with a lot of writes/reads to the data source. Also watch for updates or deletions getting out of sync with the database, causing errors in the app, this can occur frequently with apps that connect to a database over the internet when network conditions aren't ideal, or when changing networks while on the move.

A rookie mistake when updating an app for the first time is overwriting all the data the app stored on the device with the previous version. When users update the app from a store, their app data disappears.

There can be a lot of other issues such as data corruption, accidental deletion of important information, and inconsistent results when doing something like search or looking up information. This can be due to performance issues on lower powered devices, or due to new technology quirks with the persistence systems developed for mobile devices. Files or datasets might be too large, and attempting to save them causes a failure or data corruption.

Data storage may violate privacy or security laws if it is

stored incorrectly, is too easily accessible by other apps or services, or stored without the user's permission. Any data that is personal and stored without a user's permission will violate laws and device terms of service agreements.

Data can also be stored on storage cards that can be removed from the device. If they malfunction or are removed and are unavailable when the app requires the data, apps can malfunction or crash.

Advertising, In-App Purchases

Another avenue for revenue, especially for free apps, is advertising. App development frameworks provide mechanisms for this process, and there is a large industry to support ads on mobile devices. Another revenue idea is to charge for premium content, or for a more feature-rich version of the app. This requires purchasing through application stores or other mechanisms to unlock or provide more content or functionality.

Advertising and Purchase Components

To drive purchases of products or services that are available online, or in-store, ads are a powerful tool that are being developed for mobile apps. Development tools to help you create advertising within an app so that you, or 3rd parties can display ads to your users.

Games have popularized in-app purchases where you can buy all sorts of things for your characters or your world.

Other apps are looking at how successful this model is, and are looking for ways to unlock premium content, or guide users towards larger purchases from the mobile app. This depends on application store integration for purchases within an app, or integration with other third party systems to process transactions. These transactions can be complicated when it comes to licensing or revenue sharing and collection services, so they need to be thoroughly understood and tested.

How Ads and Purchases Are Used

Ads help drive users towards other products and services, whether they are related directly to an app, or merely displayed in an app on behalf of another organization. An app may be seen as one interface to an organization that helps draw people to their products and services, and it may have a lot of integration points for people to either purchase services or products themselves, or contact the organization for more information.

Ads themselves can be a source of revenue. Many free apps have ads displayed, and each advertiser pays a fee to the app developer for ad placement.

Since apps can be free, or very inexpensive, many app developers rely on in-app purchases to unlock premium content, more features, more expensive versions of the app itself or other applications to help bring in more revenue.

What Can Go Wrong with Ads and Purchases

If we want people to give us more money, and they want to give us money, it's imperative to make that process smooth and pain-free for everyone. Sadly, on mobile apps, this is often very awkward and painful, leaving someone who wants to make a purchase stuck in limbo.

Ad and purchase encouragement technology integration is new on mobile, so sometimes ads or purchase drivers can be plain old irritating to users. Sometimes you find you are unable to dismiss them, or they interfere with app usage. If ads are placed near to objects in an app that people would want to interact with first, and they load more slowly, they may load in the time it takes for your finger to gesture, inadvertently tapping on an ad, instead of what you really want to do in the app.

Ads may be a source of poor performance, especially if they use a lot of graphics or animation, effectively robbing the app or device of resources. Many ads are dependent on good networking conditions, so the apps may hang or not work at all if network/internet connections are poor or unavailable.

In many cases, store or corporate web site integration doesn't work at all. Leaving a native app and being redirected to a website for a purchase or ad can be problematic or confusing to the user if not done well. Store or even web e-commerce integration may fail completely, or crash

during the transition.

Calendar and Contacts

A powerful mobile feature of smartphones and tablets are calendar programs (so you know what to do during your day) and a list of contacts that can be used for various communications. it's fantastic to be on the move and get a quick glance at what is coming up next in your day, or to message or call a friend or colleague with last-minute information. We can conduct business or organize our personal lives while on the go using these two simple features.

Calendar and Contacts Components

The components are quite simple, the development framework we use to create apps provides both an API to access the OS-provided calendar app, and an API to access, and update your contact list.

How Calendars and Contacts Are Used

Calendars and contacts can be extremely handy to coordinate with in an app. For example, a travel app may allow you to book flights, hotels and rent vehicles, and then add these events to your calendar so you don't have to create your itinerary yourself.

Contact integration can be useful for social apps so that you can easily share information, media and images with people you are already connected to. It can also be useful to share information with people you are working with in a corporate app and environment.

Apps can access, update and create new calendar events so that you can stay up to date with your commitments. Apps can also help manage contacts, particularly within a particular context where a subset of them may be interested in what you are currently doing. Apps can also sync with other calendars and contact lists so you can share events and people with others.

An app can also provide notifications of specific events from the calendar that make sense for app usage.

What Can Go Wrong with Calendar and Contacts

These are foundational applications on a mobile device that people depend on. We don't memorize phone numbers that much anymore, we look someone up in a contacts application, and then decide how to communicate with them. We depend on calendar applications to remind us of events, and provide us with vital information on when to arrive, and what we should be prepared for. If either of these applications goes wrong, it can have an incredibly negative impact on someone's life. They miss important meetings, lose contact information for friends, family and coworkers, and end up untethered from their lives and

commitments while on the move. The last thing we want is for our app to mess up these apps!

Watch for data deletion or unwanted alteration of existing information - do not let someone's contacts go missing! Deleting or changing times/dates of calendar events could also have catastrophic consequences. What if our app prevents someone from getting to a job interview on time, or they forget a birthday or anniversary? This also extends to calendar event notifications - if they don't work properly someone might be late to a meeting or flight, or miss it altogether!

Also watch for extraneous or bogus events or contacts. Make sure the app doesn't bloat events or add in contacts that shouldn't be there. It can be incredibly irritating to suddenly find your contacts in your address book have grown and it is full of people you don't really want to contact. Now your contact list is cluttered and hard to use.

Sometimes apps can't handle a large number of calendar events, so it is important to test out the kinds of extremes that busy people may experience. Also be careful that the app doesn't overload someone's schedule with too many events or notifications that are overwhelming or irritating.

Some device operating systems may limit access to these apps, so the app may work on some devices, but not on others. Also, be careful if your apps overwrite or change default behaviors, that could cause errors.

Accessories

There is an enormous market that has grown up around smartphones and tablets, providing all kinds of things we can plug into the devices to enhance our mobile experience. Many companies supply both hardware accessories and apps to help manage them or utilize them.

Accessories Components

There are custom hardware manufacturers, and APIs in development platform to support them:

- Plug in devices:
  - Headphones
  - Microphones
  - Keyboards
  - Joysticks
  - Docks to sync with monitors, keyboards, power, etc.

There are limited options for plugging in devices, so manufacturers get creative, figuring out how to use any input in the device for their own purposes, such as headphones, power, or syncing ports.

How Accessories Are Used

Accessories can help in various ways. Input devices such as styluses, keyboards or others can help people interact with

apps more efficiently. They can enhance the experience of using media and sounds, or for creative endeavours such as video and sound recording. Apps that require a lot of typing are enhanced with keyboards or other methods of inputting. The outputs of various systems can also be utilized. Accessory manufacturers get really creative with using inputs and outputs, and each port or plug on a device is often multifunctional in both inputs and outputs. An accessory may work by plugging into a headphone jack for example, and an output may be measured by microphone activity.

- Inputs into applications:
  - Sound input adapters and microphones
  - Keyboard, mouse, stylus
  - Power cables, syncing cables
- Outputs:
  - Video and sound adapters
  - Speakers
  - Vibration motor (for games, etc)

What Can Go Wrong with Accessories

Accessories are another complication to an already complex device, and may stress a device out, particularly if it is doing a lot of other things at the time, and if an app uses a lot of resources. Here are some problems I have experienced:

- Inputs or outputs don't work, or only work intermittently
- Input causes default behavior (power, syncing with a PC, etc) and doesn't recognize accessory's alternative use off the input
- Distortion when capturing media (cheaply made input devices often pick up interference)
- Lag between inputs in the accessory and the app that is receiving them
- Poor performance in general when accessory is used.

Accessories themselves may also work poorly, so it is important to try different scenarios with the real accessory, especially failure modes such as yanking the cable, unplugging and plugging in, a loose connection, or attempting to use the app without the accessory.

Tracy's Thoughts

Tracy: It was interesting to learn about the technology used in mobile devices and what to be aware of when testing. You need to pay attention and be aware of more things than with PCs. For me, the biggest surprise was becoming aware of what sensors are and what they do. I've never really thought about most of them while using my phone before. They play a huge role in mobile, so I need to be aware of them, and make sure I utilize them when testing.

Concluding Thoughts

This section describes a handful of popular native mobile application features that you should be aware of when testing. It's a good idea to find out what features your app uses, and then correspond them with typical usage and problems to watch out for. This will help you focus your testing towards high value activities.

This is not exhaustive, so use it as a jumping off point for your own testing. Even if you are testing web apps, web sites, or hybrid apps, this is still important to understand since many of them emulate native apps, and some of the native app features can interfere with other technology if the device is working hard.

Chapter 3: Understanding Wireless Technology

"I do not think that the wireless waves I have discovered will have any practical application"
-Heinrich Herz

Thankfully, Herz couldn't have been more wrong. We use wireless waves for all kinds of mass media, and to transfer information. Wifi and cellular networks have become an important backbone in a world where we want to always be connected. Wireless technology is very different than wired technology though, and has different network speeds and strengths, and is susceptible to interference. It's important to understand how wireless works, because when it goes wrong, many apps that we depend on will malfunction, often spectacularly.

Outside of the devices, we connect to cellular networks for communications (voice, SMS, etc) and for network access, which allows us to access the world wide web. We also have wifi access, provided by wifi routers in various locations, so we can access the internet directly through an Internet Service Provider. Telecommunications companies have huge infrastructure set up with cellular towers, networks,

computing systems and other technology to support web or net access. Many devices support GPS, which relies on global positioning satellites. GPS depends on satellites in space that are orbiting earth to help us pinpoint exact locations.

Just stop and think about all of this for a second. Look at your device and imagine the paths that data takes to provide your app with an exact location, and a connection to the internet. Now think about the corresponding physical components that play a role in mobility. Sensors, antennas, computing power, electricity from a battery, etc. the list goes on.

Now imagine the wireless waves going between your device's antenna and space, and wireless waves going between your device and a cellular tower nearby, and the related hardware on your device. Now think of the systems that GPS and cellular data need to be processed by. Computers, wired networks, switching providers to cross borders, different companies, different hardware, software and languages. Does it sound complicated? It should, because it is.

Now think about what happens when you connect to a server for your app. It could have any number of technologies set up to support its network connection to the internet, and any kind of combination of server hardware and software. Then there is the program running on the server itself. Where is it located? Is it set up in an office and far away, or is it on a leased server in the cloud, at a

data center nearby? What effect might distance have on the performance of the application, particularly through wireless services that are not as powerful and don't perform as well as physical wired solutions like fibre optics? Furthermore, steel (especially in buildings) and other devices operating at similar frequency ranges can interfere with wireless signals.

Now think of the other sensors in the device that help filter out noise, re-orient and re-draw the app when you rotate the device, and movement sensors all working away. Then there is the program itself, using the computer inside the device, and it's physical components: CPU, GPU, memory and interacting with other programs and services.

Look at your device again. Now look around at what you *don't* see that powers your app experiences. Now imagine what might go wrong if any of these systems interfere with each other or if they malfunction or stop working. What effect might that have on the app you are testing?

Wireless Technology

If you aren't familiar with how mobile devices communicate with the outside world, this section is for you. I'm going to go through a brief, simplified overview of wireless technology. (If you are new to this topic, I urge you to research this more fully. It's easy, just search the web. If you understand this topic well, please don't cringe at my over simplifications.)

I want you to think back to Grandpa's radio. Remember how I mentioned that if you work with wireless people, they might refer to your mobile device as a "radio"? That's because both devices use similar technology. While you can't hold Grandpa's radio up to your ear and communicate with other people, both depend on radio waves to operate. (Note: if you want to do it for the lulz, I can't stop you.)

While you can't see them, we are surrounded by radio waves that are part of the electromagnetic radiation (EMR) that surrounds the earth. This is a form of energy that can have electric or magnetic components that bounces around in space. Like the wind, we don't see them, but they are there. Imagine poking your finger into a bowl of still water. See the waves that ripple away? Waves in the EMR spectrum work similarly, but at a much larger scale. Humans are pretty clever, we have figured out how to use this naturally occurring phenomena for many types of technology. From the humble X-Ray, to the fancy sounding gamma ray, or the waves emitted in the microwave that are burning popcorn in an office kitchen somewhere right now all take advantage of waves in the EMR.

Radio waves are used to transmit information. Grandpa's radio, or the one in your vehicle pick up these waves because someone out there set up a tower to transmit them. There are all sorts of waves out there, so you have to find that talk radio or weather network radio station by tuning into a particular frequency. A frequency is a very particular

subset of waves. If everyone used the same frequency, it would be like listening to a bunch of people trying to talk over each other. It would be incomprehensible. There is a limit to how far a radio tower transmits, so someone in a geographical area owns the rights to broadcast on that frequency, so that it works clearly and multiple broadcasts don't interfere with each other.

Have you ever listened to the radio on a long road trip in a car? Have you noticed that after you drive away from your home, at a certain point the radio station you were listening to starts to break up, fade and distort? That's because the signal from the radio transmission tower is getting weaker. As you get further away, you lose it altogether, and if you're like me, you search for another similar station that is located in the area you are now travelling in. After enough distance, you may get a completely different radio station from a different city at the same frequency as the one you started with.

Wireless technology that supports our voice communications, and our access to the internet on mobile devices works on the exact same principal. There are radio transmission towers, which also receive data, but they are usually at a smaller scale than their more powerful cousins that transmit radio stations. As you get further away from the source (the tower), the weaker the signal, and the more distortion, breakup, and interference occurs. Cellular networks are more powerful than wifi, but they aren't as fast (at least at the time of writing). Wifi have weaker signal

strength - you only need to get a few feet away from some of them to get a weak signal or lose it altogether.

Now back to Grandpa's radio for a moment.

Do you remember some days when the sound was distorted when on other days it wasn't? It just wasn't quite as clear as it should be. Grandpa would do two things - he'd adjust the frequency or address of the radio station ever so slightly. Sometimes that would do the trick. Or, he'd mutter some words he didn't want you to hear, and he'd move the antenna around. Other days, or in other rooms, that radio station would be clear and sound ideal. Why did this happen?

This occurs due to interference. The waves are moving around our atmosphere, but objects and other ways get in the way and cause problems. For example, steel can block radio waves. Weather conditions that excite electrical particles in the atmosphere can interfere with them. Topography can have an impact as well - it's easier to get a signal on top of a hill versus being in the bottom of a valley.

Have you ever noticed a local radio station distort or cut out when you are stuck in traffic next to tall buildings? That's all the steel embedded in their concrete frames blocking some of the radio waves. Have you ever noticed satellite TV cutting in and out during a thunderstorm? That's because of the heightened electrical activity in the air causing interference with wireless waves.

Our mobile devices have similar technology, but they au-

tomate the adjustment for us. We don't have to move radio antennas around or try to fine tune frequencies, they do it for us. However, they still suffer the same kinds of problems with interference. As you move around with your device, you will run into similar problems as an old fashioned radio does.

Think about wireless conditions as you test, and learn how your device displays signal strength so you can understand when it is ideal, weak, or non-existent. There are programs that you can use to help you determine signal strength, and they can help you learn about what is going on, especially if they show signal strength and your distance from a transmission source. Learn to spot sources of interference, and think about what is happening around you with regards to radio waves. When you encounter conditions that aren't ideal, incorporate them into your testing and find out how the app handles these conditions.

Wireless Communications

Wireless companies and telecommunications providers have a lot of technology to choose from. In addition to a frequency, or address that they can use, they also need to have a language that can transmit the data from one point to another quickly, efficiently and taking up the least amount of space as possible. They want to get as much out of their networks as they possibly can, and they need them to work reliably so that they can provide a service people will pay for. Wireless infrastructure handle both voice and data

communications, and they are quite complex, sophisticated systems.

Grandpa's radio station will transmit broadcasts in a spoken language. It might be English if that's what he speaks, or it might be another language - whatever Grandpa prefers and understands. Wireless systems also use languages, which are often called *protocols*. So Grandpa understands the broadcast because he tunes his radio to a frequency, and picks up a radio station spoken in a language he understands. Wireless technology breaks has frequency ranges for different technology types. You'll hear them on the news, from a salesman who is selling you a device, or you'll read it in a manual (if you are the sort that reads them.) Some of them use *channels* while others use *frequencies* to divide up the spectrum and use different technology to support our usage. Mobile technology works on similar principles.

Here are some cellular network terms you might come across (listed in order of speed):

- 4G LTE (Fourth generation of mobile telecommunications technology, Long Term Evolution.) This is one of the fastest cellular network standards available at the time of writing. "4G" Any technology that reaches this standard can use this label. Just seeing "4G" on your device doesn't mean you know what tech they are using, it just means they meet a standard.

- 3G (Third generation of mobile telecommunications technology) This is a standard. seeing "3G" on your device means your provider meets this standard and has related technology to support it.
- EDGE (Enhanced Data rates for GSM Evolution or Enhanced GPRS)
- 2G (you guessed it, second generation)

There are a lot of families of channels, frequencies and protocols for transmission, but I will let you discover those from other sources. You'll hear terms like GSM (Global System for Mobile Communications), CDMA (Code Division Multiple Access) and HSPA (High Speed Packet Access.) and RLC (Radio Link Control.) GSM and CDMA are systems that use different technology to utilize radio communication technologies. They are complex versions of the radio frequency on Grandpa's radio, sort of like the difference between AM and FM frequencies there. HSPA and RLC are protocols, or languages, sort of like the difference between a radio broadcast in French vs. English. Protocols also have speed as a factor. (Well, speed is somewhat involved with spoken languages as well, but not to the same extent.)

Note: I am simplifying things here, but this is just for you to get a basic understanding for testing.

Here are some examples to research:

- 4G: LTE, UMB, WiMAX

- 3G: CDMA, GSM, GPRS, EDGE, WCDMA, HSDPA

If you are curious, open a web browser, enter the term into your favorite search engine, and learn away. It's fascinating and complex, and even a cursory understanding will help you test better. Each of these technologies have strong points and weak points we can exploit in our testing.

These are all technology systems, and as you learn more about them, the more amazing the technology we take for granted appears. There are preferences and biases like with any technology family. For example if you have friends who are knowledgable about the wireless industry, you might find that the people who like GSM argue with people who like CDMA and vice versa. This is similar to people who love Google Android devices don't like Apple iOS mobile devices and will argue passionately with their friends over which platform is superior. Every technology family has zealous fans!

If you are using a device that supports cellular network communication, then you will see one of these terms at the top of your device when you are connected to their network. Your device will try to default to the fastest for you. So I tend to see LTE, 3G or EDGE (displayed as "E") when I am out and about. If you see the wifi signal (three or four circular bars in a fan shape), then you're connected to the web using a local wifi transmitter, and you are not connected to a cellular network.

Device Connected to Wifi (arrow points to wifi symbol)

Wifi on the other hand provides a simpler connection to the internet. As long as the person or organization with a wifi device has an account with an ISP (internet service provider) then all you need to do is connect your device to it, and it's hooked up just like a wired PC. However, it is also using technology based on waves, so it isn't as powerful or stable as a wire. Imagine a puppy chewing on the cable for your PC, and that effect is similar to what you will sometimes get with wifi. You'll probably find it is easy to find areas where you have a weak signal in your home or office. You just need a bit of distance and some interference.

In my home, the signal is weakest in the basement because of distance from the router and interference from steel framing. Wifi waves tend to go downwards, which is why

wifi routers are often set up in higher places. If they are installed on lower floors, it can be hard to get good signal strength on an upper floor. When I am in the kitchen making lunch, our microwave can interfere with wifi signals when it is operating. I can get access to our wifi router if I am my neighbor's house behind me, and across the lane, as long as I am near the back of the home, or in his back yard. I get a nice strong signal from the front step of our home, but I lose my signal completely if I am on my second floor balcony. This is puzzling at first because the wifi device is close by in my office, but I have steel framed doors that lead to the balcony and block signals. Offices are similar - the elevator is encased in steel, so you'll likely have a dead spot there. Since I know my home environment and how the wireless conditions differ, I can quickly and easily create error conditions when I move around the house.

Cellular networks and wifi use radio waves on different channels or frequencies to provide access to the web, which in turn uses the protocols TCP/IP and HTTP(S). There is a lot of technology in cellular networks to support access to the web. When I am on wifi, or a wired PC, I think simple, reasonably fast access. When I am on a cellular network, I think complexity, and I constantly think about what could adversely affect things. Even the time of day can have an affect - if a lot of people are using the same system you are, the local cellular tower you are connecting to can get saturated with traffic and slow down. There are apps that can help you check your connection speed to the cellular network and they can help enormously when

troubleshooting or logging a bug.

Another thing to note when you are testing a mobile app: energy usage. Cellular networks require more complex support, and they use up more energy on your device, so it has to work harder. That causes the battery life to go down more quickly. The faster the cellular network, the more it will wear your battery down. You'll first start noticing this extra energy through your hands - the device will heat up. Heat is a byproduct of using up energy. When a device is using more energy, an app or web site may perform poorly. For example, an app or web site may work fine on wifi, but perform poorly on a cellular network due to slower transmission speeds and the device working harder while the app is running. This is another reason why it is vital to test on **both** cellular and wifi networks. Don't get lazy and assume one will be just like the other. App behavior can be very different on one technology type versus the other.

Tracy's Thoughts

Tracy: You have to get into a different mindset when testing hand held devices due to their mobile nature. Not only do you have to be aware of what is inside the device such as sensors, but what is going on around you is important to pay attention to. Your location, physical objects that can block your wireless signal, and weather and temperature changes are not generally even considered when testing PCs but can cause major havoc to an app you are testing in the wild. Jump around, get your hands dirty, take a bus ride, go out into the rain, move around large buildings, and if you are downtown in a big city, test around sky scrapers.

Concluding Thoughts

At this point, you might be asking: "What does all of this have to do with testing? I can't even *see* what Jonathan is talking about. Why is this important?"

First of all, cellular networks are complex and variable, and the timing of communications can vary a great deal. When technology is this complex, there are is a lot more opportunity for problems. When apps depend on access to the web to work, they may not handle poor connections, or moving from one type of network to another all that well.

In fact, at the time of writing, this is one of the most problematic areas I have found when testing apps. Many app developers assume a strong, consistent, singular network connection, and don't put in error handling to deal with these issues, so their apps just crash when you move around and they encounter weaker signals or cellular network changes. If required data arrives too quickly, out of order, or too slowly, an app may malfunction or crash, just due to timing differences from one network type to another.

It's also important to understand that the devices work really hard for us when they use wireless networks, which means they use a lot resources on the devices themselves. When you have a nice strong wifi connection, the device doesn't have to work very hard. But as you move around, it is trying to find the best network, and might be changing channels, adjusting to a new connection, trying to filter out interference and all sorts of things.

This taxes your device: your CPU (central processing unit, the heart of the computer), memory usage, inputs and outputs, and wireless optimization work harder as these different programs and services try to adjust to give us the best performance. That means the program you are testing has to compete with these other resources, and is no longer working in an optimal state. If our program is a bit of a resource hog itself, that can cause a lot of problems with performance and reliability.

Finally, having a basic understanding of how wireless mobile technology works helps you determine the cause

of an intermittent bug. If you know nothing about cellular networks, wifi, sensors, batteries and other things our devices depend on, you'll log a bug and if it can't be repeated by others, it won't get fixed. If you notice intermittence, and take notice of some of these factors, you will be able to narrow it down and get a repeatable case that can be fixed. Then you'll have an internal monologue like this: "Weird. When my device moves to EDGE network when I have a low battery, our app fails, but it works everywhere else."

Chapter 4: Mobile Testing Tours

"That Led Zeppelin biography Hammer of the Gods was like their tour guide." - The Captain, describing being on tour with the Beastie Boys

Cem Kaner has described testing tours as a directed search through a program. Testing Tours: Research for Best Practices?[3] They have become quite popular testing approaches, even showing up at companies like Microsoft and Google. The Touring Test[4] The FedEx Tour[5]

I was introduced to the concept of tours by James Bach. James is a software testing consultant and trainer, and we were spending the day together sharing ideas and working on skills and techniques. I had just tested a simple application and had explained my observations and discoveries. James took over and demonstrated how he would test a new application.

"I'm going to start out with a feature tour."

---

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

[4] http://blogs.msdn.com/b/james_whittaker/archive/2009/02/12/the-touring-test.aspx

[5] http://googletesting.blogspot.ca/2009/10/fedex-tour.html

I then watched in amazement as James quickly and systematically explored the application from top to bottom. He showed me things I hadn't noticed in my previous test session, and there were entire features that I didn't even know existed in the application. "I want to learn how to do that!"

James explained that he just started from the top left of the application and worked his way to the right, then down, left to right, until he had explored a screen in its entirety. He would then move on to other screens, and systematically explore each one. Not only did the tour help inform him of what the application was capable of, and help him think of areas to test later on, he also came across bugs as he worked his way through the application. My previous work was not as systematic and thorough, so I didn't have as many observations, bug discoveries or potential problem areas to explore.

Two tours I learned from James that I use on mobile apps a great deal are the feature tour, as I described above and the user tour. The way I implement a user tour on a mobile device is to simply try the application out myself, being careful to note any crashes, instability or usability problems. I then think of other credible users, and try to imagine how they would use the app. I then pretend I am one of those people, and use the app from what I imagine their perspective might be. James has identified several tours that he has shared with testers, and I have bolstered these with some that I have found particularly useful for

mobile devices.

With mobile devices, we are not only testing software, but we are testing how it works when we are on the move, depending on different network types and location services, and how the hardware and sensors within the device are utilized by the software. When we test the devices taking physical hardware, the environment we are in, and contexts where we use the software, it is interesting to see how they all interact together. On PCs, testing tours are often a metaphor, a powerful thinking device. On mobile devices, since we need to test them on the move, tours are also a real, physical thing.

To use these tours ideas to help guide your testing, you can use them as fact-finding missions to help guide your testing. Or, if you are like me, you can just use tours to help frame your testing and discover new and important bugs.

Gesture tour

Go to each screen in the app and try every gesture you can think of. Use them where they make sense, but also try to gesture where there are no visual cues. Since mobile apps have very little help text due to a lack of space, app designers expect people to start inputting gestures and exploring the application to learn how to use it. Usually we stumble on inputs that yield features, as we work our way through an application intuitively, but using a

systematic approach will reduce trial and error, and speed up observation.

How to Do It

On every screen of the application, try out each one of these gestures. Try them on each object you see such as buttons, sliders, pickers, links and text fields, but also try them in white space or in areas where it isn't obvious that there is something to interact with.

Gestures:

- **Tap** (push your finger tip on the screen and let go right away in a tapping motion) Used to start applications, click on buttons, links, etc.
- **Double Tap** (push your finger tip on the screen twice, rapidly) Used to rapidly zoom in or out on content, images and animation.
- **Press** (push your finger tip on the screen and leave it there) Used to operate some features, such as a home screen button.
- **Press and Drag** (push your finger tip on the screen, then move your finger tip on the screen in any direction) Used to move an object on the screen, or move your orientation in different directions at once. (Note: as John Carpenter points out, this gesture combination can be supported differently on different types of devices. On some, you can combine

the movements, while on others you have to press, release then drag.)

- **Swipe** (slide your finger tip across the screen quickly, in any direction) Used to lock and unlock screens, to move or dismiss items
- **Flick** (very quickly slide your finger tip in a rapid motion in any direction) Used to quickly move or dismiss items
- **Shake** (physically shake the entire device) Often used to clear input fields or reset an application to a default state
- **Rotate** (press your finger tip on the screen and draw a circle on the screen) Used to change perspective or view, or as an input for a game or entertainment
- **Pinch** (press two finger tips on the screen, then bring both of them together on the screen in a pinching motion) Used to zoom in on an object on the screen
- **Spread** (press two finger tips on the screen, and move them apart in a spread motion) Used to zoom out on an object on the screen

Also include any off-device motion gestures that don't require a touchscreen if the device supports them.

What to Watch For

You will quickly learn about different features that are available in this application by doing a systematic tour

of every screen and inputting each gesture type. You will find that some of them launch features that you may not have been aware of, and in some cases, a gesture will do absolutely nothing.

You will also find problems with consistency from screen-to-screen. On one screen, certain gestures implement certain features, but on another screen, the same gesture(s) might implement completely different features, which is confusing.

In many cases, you will also find that some gestures are poorly supported, and the inputs lag or screens don't redraw properly after entering some gesture inputs. In some cases, certain gestures may cause the application to get into an error state or crash. Sometimes, due to the use of third party libraries when programming an application, a programmer may not realize that some gestures are supported in the application when they should not be. Inputting gestures that the developers didn't intend for the program to use can have unintended consequences in the application.

On hybrid apps, this is much more common because there is more work for the programmer to put in code to handle different gestures. They may not know about some gestures, or may forget to test for gestures they weren't thinking about during development.

Touch targets (small areas on the screen that respond to gesture inputs) may be too small, or too large, causing usability issues. If the touch targets are too large, then it

may be difficult to interact with other objects next to an input area, such as zooming in and out. If the touch targets are too small, or offset from where you think they should be, it may be difficult to consistently get the application to accept inputs and access to features or text inputs may be difficult to do well.

Get creative with gestures and incorporate movement as well. Also realize that people may use devices in ways that the programmers haven't intended. Tracy created the "toeswipe test" because she uses her toes to interact with a tablet that is placed on the floor in front of her when both hands are busy, such as when she is knitting. There are a lot of situations where people may get creative when they are using a mobile device when their hands are busy with something else.

Orientation Tour

Work your way through the application in portrait view (device held up and down), then rotate the device on its side and work through each screen of the app in landscape mode.

How to Do It

Variation one: work through each screen in portrait, then turn the device on its side and try to repeat in landscape mode. Variation two: change orientation on each screen.

Try in portrait, rotate, try in landscape, then rotate back to portrait, move to the next screen.

What to Watch For

Some applications look better in one view over another, so they may limit how the application can be viewed. If this seems to be the case, watch for any inconsistencies: can you rotate the device on one screen causing the application to be less usable? This might mean that the orientation should not be permitted on that screen.

It can be easier to enter in a lot of text on a device in landscape mode. On screens where there is a lot of typing, such as a notes page, does the application support landscape? If not, it probably should.

Watch for inconsistency in supporting both views. Some parts of the app may support an alternate view, with the odd screen only supporting one. Supporting both portrait and landscape mode in an application can be a lot of work, so the development team may take shortcuts. An application might launch in portrait mode, and work when you turn it to landscape, but when you are in a landscape view and try to enter in a text or pick an entry from a picker, the screen may revert to portrait and not support landscape. This can be an irritating inconsistency.

Change Your Mind Tour

Since mobile applications tend to have a singular focus with a theme on each screen, it can take more effort to get into a workflow that a user might commonly work through as they use the application. For example, it may take several screens to get into an area of the application that might be shown on one screen in a web or standalone application. When we design and develop applications, we often focus on happy path scenarios and don't focus enough on scenarios where a user might change their mind and want to go back.

How to Do It

Use a systematic approach:

Go Back a Screen When you go from one screen to the next screen in an app, try to go back one screen.

Get Me Out Navigate through the application, then imagine a user is in an area they do not want to be in, try to go back to the beginning easily.

Cancel! Work your way through the application, and if you encounter a screen where you need to enter information to submit to a system, make a purchase, etc, try to cancel and go back to a prior screen, or cancel altogether and get back to the beginning.

Close and Re-open If the application loads an image or other media viewer, close the application and start it up

again. If it opens the application in the media viewer, you may not realize that the application has started, you may get confused about just what has launched.

Force App to Background, Reopen You can send an app to the background and relaunch it, where it should be in the same state as it was when you sent it to the background. You can also relaunch if it has a notification that appears while you are using another app. You will have to review your device's help information to figure out how to do this. (Thanks John Carpenter for pointing out this combination.)

What to Watch For

Be on the lookout for awkward workflows, or the complete inability to return to a starting page or within an application. This can be incredibly frustrating, especially if a user is trying to check on important information and ends up in the wrong place, or accidentally initiates a social connection, purchase request or another action they would like to quickly cancel and move back from. App designers should be aware that users may close the application at any time, and re-open it. The way the app launches after re-opening it can be a big source of confusion if the ability to go back isn't anticipated in the design.

Accessories Tour

Mobile devices support a wide variety of inputs from accessories such as headphones, microphones, power ca-

bles, device cables and input devices such as keyboards or joysticks. Most touchscreens have accessories for inputs as well, such as a stylus or special gloves for inputs as well as portable keyboards and pointing devices.

How to Do It

While using your application, try the following with physical accessories:

- Plug the device into a power outlet to charge it
- Plug the device into a PC
- Plug in or remove headphones
- Plug in a microphone
- Plug in other devices
- Interact with the device you plugged in while using the app
- Jiggle the cable while using the app
- Try every gesture with a stylus or glove

Wireless accessories:

- Sync with a keyboard
- Sync with a display device
- Control another device with your handset (like a TV remote)
- Try the accessories around monitors, microwaves or other devices that emit frequencies that can interfere

- Try the accessory while moving your device farther and farther away to see how it works with weak signals

Reader and mobile tester Alexander Khozya recommends combining activities while using accessories:

- Rotate the device to redraw the screen from portrait to landscape while plugging in or unplugging a device
- Move the device around while wiggling the cable, or plugging it in or unplugging it
- Incorporate a lot of movement while also using an accessory to stress the device out

Utilizing device features while interacting with or plugging/unplugging accessories is a great way to find problems that other people may not have thought to design or test for.

What to Watch For

Accessory usage causes sensors to be activated, and for some other processes on the mobile device to come into use. This can cause performance problems in the application you are using. In other cases, plugging in an accessory may cause an interruption from an alert or message box that the programmers hadn't thought of that can cause the application to freeze up. Error handling for accidental or

unforeseen accessory interaction may be non-existent or very poor.

Many accessories aren't made particularly well, and they can cause noise to interrupt the application. If the application depends on media such as music or sound, they may interrupt recording or playback due to sound triggers, or static created by unshielded cables. For example, a recording microphone may not have shielded cable, so when you jiggle it, the electricity your body gives off may interfere with the input signal in the device, which causes a static noise.

Wireless accessories may not work well if there is interference or a weak signal. Try them in different locations to see if some contexts are better or worse than others. The target context may not work very well at all when you try it under real world conditions.

Motion Tour

Smartphones and tablets have sensors that are used to detect movement of the device. These can be used in a variety of ways in an application. Inadvertent movement, or actions that cause several sensors to work at once may cause applications to behave strangely. Accidental or exaggerated movement is common when we use the devices on the move. For example, if we are using a device on a train, we may be subject to sudden movements when the train stops or starts or goes around a tight curve.

How to Do It

Hold your device tightly while the application you are testing is open. Try the following movements while holding the device with application open:

- Move your arm in a circular motion,
- Move your arm back and forth (easiest to try when walking)
- Rotate the device in circles while moving your arm: get it to rotate, but also move the device so that the screen isn't activated
- Move the device and stop suddenly
- Combine movements: move your arm while rotating the device from one side to the other

What to Watch For

Movement can trigger several sensors at once while your app is also working, which can cause performance issues. At worst the device might lock up, causing the dreaded gimbal lock. Applications can crash due to too much sensor input without enough error handling. If the application uses a shake gesture, movement may cause inadvertent actions in the app that the user didn't intend. For example, if a user is making a purchase using an app while on a train, a sudden stop might cause the shake gesture to cancel their order they just filled out. If the application provides access to content, and a shake gesture takes you to a random page,

you might end up on a page you don't want to see, or get confused about where you are in the app.

Light Tour

Mobile devices have a sensor to detect ambient light, and to lighten or darken the screen depending on what the surrounding light conditions are. If ambient light is low, or you are in darkness, the screen brightens, and if the ambient light is bright, the screen darkens automatically. Since the devices are mobile, they will be used under all kinds of lighting conditions, so the application design and colors may need to adjust if they can't be used under all kinds of light conditions.

Light sensors also work with proximity sensors to determine whether to lighten, darken, or turn a screen off. A proximity sensor senses when you (or one of your appendages) are nearby.

How to Do It

Try the application in different lights, from different sources:

Inside:

- Normal artificial light (lights are turned on in the room)
- In a dark room (a room with no lights on or windows, like a closet)

- Under or next to a bright light (sit next to a lamp)
- Also try variations of light sources, such as fluorescent or other kinds of lights
- Cover the device face, or parts of it to see if it triggers changes due to a change in light

Outside:

- Regular sunlight
- Very bright sunlight
- Cloudy or overcast conditions
- Dusk/sunset
- At night, in the darkness

What to Watch For

The application design may not work well in different lighting conditions. It may be difficult to see, and therefore difficult to use or interact with. The color of the background of the application may be too bright or dark to adjust to different lighting conditions, as well as any application controls such as buttons or text fields, as well as any images. User interfaces tend to be developed and tested inside offices where we are using PCs or workstations, so the designers and programmers may never think to try the application under different lights. Also watch for eye strain under certain light conditions. If the design of the app is difficult to look at under certain lighting, it may require color changes to make it more usable.

Some colors may not work well for people who are color blind. In some cases, they may be ok under certain lighting conditions, but with brighter outside light, the auto-adjustments the device makes may cause color blind people to not be able to view app features or information on the screen.

Location Tour:

Location services are a huge part of mobility applications. Applications take advantage of knowing where we are to provide better service. When they work well, they are a powerful tool that make mobile apps truly mobile. When they go wrong, they are frustrating and render our apps that depend on providing our current location completely useless.

How to Do It

Get out and move around. The following activities can make location services difficult to work properly: Walk at a good pace, then stop suddenly. Location services that are calculating while on the move anticipate where you will move next. A sudden stop can cause them to mess up calculations, especially if you repeat this over and over. Also, just stay in one location for a few minutes and monitor your current location. Does it change or stay the same?

Walk near tall buildings. Radio waves can't go through steel, and GPS services require a clear path to satellites, so if there is steel and other obstructions (common when using an app in a city) location services can get confused about your exact location.

Use the services during busy times during the day. Some location services depend on radio towers for location determination. If a lot of people are on the system, the tower can be saturated with traffic and there can be delays in determining your location.

Use the services during different kinds of weather. GPS can be affected by thunderstorms.

Try location services when you are inside, and in different buildings. Steel and geolocation information can cause strange results when you are inside a building vs. outside of it.

What to Watch For

Location services can be completely wrong, depending on factors such as interference, time of day, the traffic on radio towers, being inside vs. being outside, inclement weather and others. It can also be inaccurate. For some apps, you don't need a lot of accuracy, but for others, pinpoint accuracy is important. For example, if you are trying to find your way in a strange city and your location service is accurate at about a kilometre distance, that application will be unusable.

Many apps depend on location services to function. What happens if you turn off location services, or set your device to airplane mode and try to launch the app? Does it crash or does it provide you with appropriate error messaging that is helpful?

Connectivity Tour:

Network connections are nothing new with applications, but in the past, we depended on reliable, stable, constant connections to external sources of information. With mobile, this is no longer the case. Since we move around with the devices, we depend on different technologies to connect us to the internet or other sources. At home or the office, we usually use wifi, but when we are on the move, we depend on wireless broadband services through mobile providers.

How to Do It

You will need to explore the areas where you use your device, at home, at a coffee shop, the local library, etc. to find wifi sources. Try using the application and move from one type of connection to another:

- Strong wifi (3 or 4 bars depending on the device) to weaker (1 bar, looks like a dot)
- Move from Wifi to a cellular network
- Move from a cellular network to wifi

- From one wifi source to another
- From one cell network type to another:
  - 4G to 3G
  - 3G to EDGE
- From connected to no connection (moving into a dead spot)
- Move from a dead spot to a wireless network connection

What to Watch For

Applications that depend on information and data from the internet or other sources often don't handle transitions from one wireless source to another very well at all. They can hang, crash, or go into strange error states. They also don't do very well if the connection isn't very good, if it is weak, or using a slower wireless broadband technology like 2G or EDGE.

Weather Tour

Mobile applications are used in all sorts of conditions, and that includes different sorts of weather. Location services, light sensors and wireless communications are all affected by the weather, so try to combine weather tours with others.

How to Do It

Try the application outside, in different weather conditions. Ideally you will be able to try different temperatures, and different lighting and other phenomena such as rain, snow, wind, etc.

What to Watch For

Cold weather may affect sensors in the device, so app developers may need to adjust how the application uses sensors if they don't work well in the cold, if an app needs to be used in cold weather. Also, in snow and cold, users may have gloves on to protect their skin, so if they need to take off gloves to use the application via the touch screen, make sure that workflows are short and quick, so that skin exposure is minimized.

Inclement weather can require usability to be much more polished, because a user is highly motivated to quickly use the application and move on. They don't want to be stuck dealing with weather and fighting with an app that is slow to provide them with what they need, further raising anxiety or frustration. Inclement weather tends to put us in a bad mood, so we are less forgiving of app usability issues.

Different weather also provides different lighting conditions, so it may be difficult to see and interact with an application under different weather conditions. For example, an app that works fine in sunlight outside in the summer

may not work well with the extra bright glare provided by snow on the ground on a sunny day.

Comparison Tour

Many of us have more than one device, often we have a smartphone and a tablet or e-reader. Each of these have different screen sizes, which is notoriously difficult to support with mobile applications.

How to Do It

Take two (or more devices) that have different screen sizes and if possible, different operating systems. For example, you may have an iPhone or Windows Phone smartphone, and an iPad tablet or an Android-powered e-reader that allows you to install and run apps. Take a systematic tour on both devices at once, one screen, one gesture, one feature at a time and compare.

Also try different OS versions on the same device type. New versions often have different implementations or new features, and may introduce new problems. New OS's come out frequently and quickly, and sometimes they break something that worked before.

What to Watch For

Differences:

- Layout and design issues. The app might be optimized for a smaller or larger screen, and work better on one size that another.
- Missing features. One device may show a menu bar, or display different buttons and options than on another.
- Inconsistent gestures and workflows. The app may work well on one device family (often the favored platform of the development team) and may not work as well on another.

This is a difficult area to develop applications for, because different device types have very different implementations of gestures, layout and design and interactivity. In some cases, devices may not support features that are common on other device types. Our job is to report what we see, and let designers and developers figure out what to do when they hit technology constraints. Inform, but don't always expect perfection from the developers and designers.

Sound Tour

Applications that utilize media depend on sound playback through speakers, and in some cases, sound recording and editing, requiring microphones or other equipment to pick up sound and process signals. Many devices also produce sound, whether from built-in applications such as telephones, text messaging, error messages or from other

applications. Sounds, and the processing required to create them can interfere with different kinds of applications.

How to Do It

Media Apps:

- Generate sounds while using the application using features within the application
- Pay attention to sound recording, playback, etc. features and analyze how they work

Other Apps:

- Interrupt another action with a sound:
- Generate a sound from the phone (with a phone call, or a message or from another application while uploading files, doing a search entering information, or transitioning from one screen to another
- Generate sound while inputting into the application with a gesture, or a screen rotation

What to Watch For

Be aware of poor performance if music or media is playing while you are performing another action with an application. Listen for distortion if you are using the app while recording or editing sound files. Apps should not play

sounds when your device needs things to be quiet, such as during voice or video communication.

How does your application handle interruptions such as phone calls or error alerts or beeps? Do they get recorded in your sound file, or do they cause distortion? Do sound interruptions in an application cause other actions to get interrupted, error out, freeze up or crash?

Combination Tour

This is adapted from one of James Bach's tours. Mobile devices and the applications they support utilize multiple technology types, such as software, hardware, device sensors, network connections to the internet, location services and communication. While each of these aspects have their own set of strengths and weaknesses, the real power in mobile applications are the emergent properties that occur when we combine technologies within an application experience. We also combine activities when we use the devices: we are doing something else and we use an application on our mobile device.

How to Do It

Cause multiple technologies to work together on the device:

- Move the device while interacting with the application using gestures or inputting text, causing touch

screen sensors, gesture interpretation and movement sensors to work together.

- If the application requires a web connection, click a link or submit text and then move the device. Network requires work and processing, so adding in other activity when the application is trying to connect to a server on the internet adds more processing overhead. To determine when the app is connecting to the internet, click links or submit text, and watch for a processing wheel to appear. When you can easily cause web requests, then add in movement.
- Repeat the above, such as gestures or text inputs or web requests while moving between network connection types.
- Use the device while doing something else:
  - Use it while watching TV
  - Try the app while out for a walk
  - Use the application when preparing a meal
  - Try anything else you can think of. Make it realistic.

What to Watch For

When you force different technologies to work together, be prepared for unintended behavior in the application. For example an application may hang, crash or simply perform poorly if too many sensors are being utilized at once while it is trying to interpret input gestures.

Network connections to the internet can utilize a lot of resources, so causing other features on the device to be activated at the same time may cause connection failures or performance problems. When you use a device on the move, location services and sensors may also be activated, which are also intensive and may steal resources away from what you are working on at the time.

Network changes are a difficult area to program well. We often assume a reliable, strong internet connection, and we don't think about the volatile and dynamic nature of web connectivity on the move. Be prepared for usability issues (I lost connection, so I can't access what I need, right now!) or for application crashes.

Consistency Tour

This is another tour that I learned from James Bach. Mobile applications are often rushed to market, and utilize a lot of 3rd party libraries. As a result, the applications are often not consistent from screen to screen, or feature to feature.

How to Do It

Work your way through the first screen of an app, incorporating a feature tour and gesture tour. Make note of each feature and what each gesture does. Go from screen to screen, and note any differences.

Review the application developer guidelines for the operating system you are testing. Watch for anything that is inconsistent with the common or recommended practices for that particular OS.

What to Watch For

This can result in apps that are very confusing to use. For example, if a double tap does one thing on one screen, and something completely different on another, it can be simple to get lost. One application that I have used has an inconsistency with sensor interaction. If you rotate or tilt in one view, you get a 3D map view of your data. If you do the same gesture in another view, it initiates a phone call. I have inadvertently made phone calls to different organizations, which can be quite frustrating, especially if it is long distance.

User Tour

This is another tour adapted from James Bach.

Usability tests with real, live users are incredibly powerful tools, but many of us don't have the budgets or resources to utilize them. I've found that by being creative, and doing a bit of pretending, we can get some pretty good results on our own. Real users are the best, so try to find some people outside of your development team to help, but also try out user tours on your own. Hey, who doesn't like to behave like a kid and pretend once in a while?

Imagine you are a user who has far less capabilities on a device than you do. Is there a family member, child, spouse or friend that you can think of who struggles with technology? Now think of someone else who uses technology differently because they find it easier to use than you do. What are the key differences between how those two different people use mobile devices and how you do?

If you are having trouble, go to a local coffee shop and observe other people as they use the devices. Do they get frustrated? Talk to them (without being creepy) and ask them about their devices, what they like and don't like. Watch for emotion: are they happy? Angry? Frustrated? Does technology make them feel dumb?

How to Do It

It's important to try to get inside the head of the person whose perspective you are testing from. Actually pretend you are the user. You will know you are getting it right when you use an application from their perspective and you notice different things about it than when you used it before.

Technophobe User:

Do you have a family member who always asks you to fix their computer or their mobile device? Do you spend family get-togethers fixing their computer instead of visiting with

your relatives? What are their most common questions and struggles? Now imagine what they would ask you if they used the application, and try it out. This is sometimes called the "Mom test."

Child User:

Do you or a friend or family member have a small child? Have you noticed how they explore devices? They tend to be very physical, just like with any toy they learn about and interact with. They are comfortable tapping, gesturing and moving the device around to learn about what it does. They will rapidly learn by using extreme gestures, or doing something over and over. They will also add a lot of variability into their actions to quickly learn about what they are interacting with. They will also get bored easily, especially if there is a lot of text, and not enough action coming back to them from the device.

Teenage User:

Have you ever watched a teenager or young person use a mobile device? They are very quick with their typing and interactions, and they context switch constantly. If they load an app and it is slow, they will check email, text a friend and load a social networking app, then switch back. They tend to content snack and rapidly switch between usage between several apps or mobile functions. This can be very difficult for apps to handle.

What to Watch For

This tour is a treasure trove of usability issue discovery. Your aim is to expand your usage beyond your regular habits and style of interaction. Mobile devices are used by children, senior citizens, and everyone in-between. Being able to observe traits and habits as well as skill levels of different users, then mimicking with your own testing is a powerful approach. You will be surprised at the problems you find related to user confusion, awkward workflows, and poor application performance, just by using it differently. You will also probably discover features of the application you didn't know existed, and in some cases, that perception switch will reveal bugs that you didn't notice before.

Lounge Tour

This tour is inspired by our friend Tracy Lewis. Tracy described testing apps in a typical use around the home: lounging on comfortable furniture or in bed. How often do we use mobile apps when we are relaxing on a couch, or lying down in bed? We also use productivity applications when we are sick in bed, or to get in some last minute work before we go to sleep. We also tend to read in bed a tremendous amount.

How to Do It

- Try the application while lying down on a couch, or a comfortable chair.
- Use the application while lying in bed.

What to Watch For

We interact differently with the devices when we are lying down. We gesture differently, and we tend to hold the devices in a way that is comfortable for us to read, with our arms propped so we don't get muscle strain. This often means the device is held at an angle, which can cause the sensor that determines portrait or landscape views to get confused. Apps may constantly switch from one view to another over and over just when you get your arm into a comfortable position. You may discover missing features, such as an orientation lock that would prevent constant portrait/landscape refreshing. You may also find that touch targets are the wrong size, gestures get misinterpreted, or that the layout of the application is difficult to see when lying down.

Low Battery Tour

The devices work differently when batteries are getting low on power. The devices may have built-in power saving mode that disables features, and other applications may also try to manage their battery use by turning off certain

kinds of networking, or other activities that drain the battery, which could negatively impact the app you are testing. Watch your battery life, until it gets close to your first warning, then start using the app.

How to Do It

- Use the app while the battery is low, until it dies completely
- See how your app handles interruptions from battery warning messages
- Once the battery dies, plug in a power cable and see how the app handles this interruption and change in state

What to Watch For

- The app may depend on features that get turned off when the battery life goes down, so it may crash or freeze if there are changes in the system to features it depends on.
- The app may suffer from performance issues due to some resources being turned down or off.
- Other apps may turn off features your app depends on, causing crashes or other problems.
- Other apps may have opposite power optimization features - while your app turns off a feature, another may turn it on, and they may completely use up all your processing power as they compete for CPU and memory resources in an endless loop.

- Changes in state may occur with the device when a power cable is plugged in, and it may use resources as it is charging, and not perform as well.

Temperature Tour

Heat

Another area where devices go into optimization mode is when they heat up. Heat is a byproduct of energy usage, and certain tasks and activities cause the devices to get warm. When they are quite warm (not burning you to the touch) many device systems will start to initiate activities on the devices to help reduce energy consumption. They may slow down CPU clocks to reduce energy, free up memory, shut down unused services, or turn off services that use a lot of resources.

Lighting up the screen is a big resource user, as well as utilizing cellular networks. Have you ever noticed your device will heat up quickly if you are on a cell network rather than wifi? Also, using devices outside, or in direct sunlight can cause them to heat up. Another factor that can cause heat is plugging them in to charge the battery.

Cold

Many climates have cold temperatures for several months of the year, and devices need to be used outside. If you have numb fingers, or want to use the device as little as possible because it is cold, that has a huge impact on usability and performance impressions.

How to Do It

Note: Be careful with this tour! Do not willfully damage a device!

Heat:

When you notice that your device is hot or warm, try out a tour through the application and pay careful attention to any problems that might occur.

Cold:

Try out the app if you are in colder weather. Can you interact with cold or slightly numb fingers? Is the app fast enough to let you get your gloves or mitts back on?

What to Watch For

- Poor performance (the device OS might slow down processing if it is hot)
- Crashes or strange errors (the OS might turn off networking, or may dim the screen, or other services to try to reduce energy consumption)
- Usability issues in colder weather may become much more important, especially simple inputs and good performance

Multi-Screen Tour

Mobile device usage is often combined with other devices, or "screens" such as PCs, laptops, gaming consoles and

televisions. If your app can be used in conjunction with another device, such as a web application or site on a PC, or with a television-related service, then this tour is for you.

As Google Analytics point out, there are two main categories of usage: *simultaneous screening* and *sequential screening*. The New Multi-Screen World[6]

Simultaneous screening occurs in parallel. You are doing one thing on one device, such as working on a PC or watching TV, and you do something else with your mobile device, such as text or check email. In many cases, we supplement or augment the experience from one screen with our mobile device. We may take part in an online discussion or social media while watching TV, or look up information about a movie. We may also research a particular organization on a PC while we are using their app on a device.

Sequential screening involves using different devices to complete aspects of a task one one device, and finish off on another. Shopping is one example. I may search for available airline flights on my mobile device, then book the flight and pay for it on a PC (since there is more typing and it's easier that way), then utilize the ticket, and keep tabs on flight information on my mobile device.

How to Do It

This requires multiple devices for testing.

[6] http://services.google.com/fh/files/misc/multiscreenworld_final.pdf

- If your app could be used at the same time as other products or services on another device, then use your app in parallel, and simultaneously to other products and services on other devices
- If your app can be used to solve a problem and reach a goal on multiple devices, then test a workflow by moving between devices to try to accomplish one task.

What to Watch For

Simultaneous:

- Key information might be missing on a mobile version, and it might be difficult to find on another device.
- An app might not be usable if your attention is split between multiple devices. There might be time-related notifications that you miss, or the look and feel and interactions might require all of your concentration.

Sequential:

- You may not be able to complete tasks using more than one device, you may be forced to do the entire workflow on one or the other.

- Watch for sub-tasks that can't be completed individually on a device, such as search on one, make a purchase on another.
- Many systems do not sync together well, so check that changes you made on one device are accessible and reflected on another.

Pressed-for-Resources Tour

Try to create conditions on your device so that it has to work harder while you use your app. There can be bottlenecks or performance dependencies that are difficult to spot under ideal conditions.

(John Carpenter and Johan Hoberg asked me to include this tour, and it's an important one for finding out interesting information. I initially called it "performance", but Johan supplied a much better name. Thanks Johan!)

How to Do It

Tour through the app under ideal circumstances: a fast wifi network connection, ideal lighting, and without moving the device.

Next, repeat, but try stress the device more so that it has to work harder to support your app. There are a lot of variations that you can try one at a time, and in combinations:

- Tour the app with as many other apps running at the same time as possible, even if they are in the background. This will cause competition for resources and memory.
- Fill up internal and SD card storage (if your device has one) with media files, and see how the device responds.
- Use the app with a slower cellular network such as EDGE or 3G vs. a faster wifi
- Try the app with weaker cellular or wifi signals, which makes the device work harder, particularly if you are in a transition point between two network services
- Play music while using the app
- Incorporate gestures while you do something else, such as shaking or rotating the device while inputting or gesturing, or when submitting something to the web
- Use cellular networks during busy times of the day, such as over lunch hour. Local cellular towers may get saturated at certain times of the day, so see how your app handles poor network performance.
- Use the app when the battery is low, when it is charging, or when it is hot, to activate any optimization features on the device that can slow things down.

Pay close attention to what combination of factors cause performance-related issues. Take note of what wireless network you are using, signal strength, the heat of the device,

battery life, and what specific movements and activities you undertook.

What to Watch For

Performance issues can occur much more readily, or manifest themselves much more obviously when we add in factors that are more real-world to our testing. People jostle and shake devices while moving and interacting with a device on a busy cellular network for example.

Watch for apps hanging, crashing and not responding well to working while the device is stressed, even if it is for a second or two. They may go into endless loops retrying to connect to a server.

Apps can and should have error messages to let you know what is going wrong so you can try to fix the issue and use the app again. For example, I have an airline app that detects low memory on the device, and asks me to shut down other applications and try again. This is a brilliant usability feature that can help me figure out what is wrong, fix it, and then use the app as needed.

Emotions Tour

This is similar to the user tour, but I have also found that emotions play an enormous role in mobile device usage. Since they are embedded or enmeshed in our lives, we take them everywhere. We take them with us into the restroom

to help pass the time as we answer the call of nature. We take them to the bank, when we go shopping, and to special occasions.We may be in a church for weddings, funerals and services, or at other gatherings such as plays, concerts, dances and sporting events. We have them in classes, in meetings, at work, and in social gatherings. Many of us also sleep with them beside our beds so we can access them if we wake up during the night. We also take them when we travel away from home to other destinations.

While we go through any of those situations, we have different emotions depending on the context. Our emotions have an enormous influence on how usable or useful we find an app. If we are angry at something else, we may get even more angry if the app is frustrating. If we are sad and depressed, and the app crashes, we may take it personally and feel even more sad. If we are fearful or anxious, we want apps to respond and perform quickly to help us alleviate fear and anxiety.

How to Do It

Either watch your emotions, and take them into account for this tour when you are feeling a certain way, or try to simulate them. Get creative and try to imagine how you feel in different situations.

Try the app when you are feeling the following:

- Happy

- Sad
- Depressed
- Anxious
- Fearful
- Angry
- At peace
- Indifferent
- Frustrated
- Joyful

Pay close attention to how you move, behave and think when you feel those emotions, and pay close attention to how you use the app differently, and how your expectations of the app change. Note anything about the app that makes things worse, or better.

What to Watch For

Watch for usability and performance issues suddenly becoming apparent when you are feeling a different emotion.

Imagine you are using a banking app. It may be usable if you are happy, sad, angry or at peace. But what happens if you feel anxiety or fear? If your bank calls you to tell you that there is a problem with your account because there is suspicious activity and asks you to review your account information, how will you feel when you use the app?

Physical changes occur in the body when we get news like this. Heart rate goes up, and we often shake. We are

impatient and in a rush, so we may press much harder on the screen as we gesture. We may even hold the device completely differently - are we squeezing it and holding it at a different angle? So now we are engaging sensors in a way we normally don't. We also tend to sweat when we are anxious and fearful, so we may not be able to have the fine motor control when trying to input into the device, or the sweat may cause our fingers to slide around more, making inputs more difficult.

Now take time into account. If I'm relaxed, waiting a few seconds for an app or screen to load isn't a big deal. If I'm anxious, suddenly every second feels like minutes, and I become even more anxious. Performance is now much more important - I need to find out NOW what is going on.

Also take reliability into account. If the app crashes when I try to check my balance, or has an error, I may go into full panic mode. This is a nightmare scenario for an app designer - our app actually increases potential harm rather than reducing it.

I'm in a Rush! Tour

This is similar to the emotions tour, since they are related. Also take location and weather into account, since we are often in a rush in different locations under different conditions.

We may use apps completely differently when we are in a hurry. Our expectations and usage can be very different,

and this type of tour provides vital usability information we may miss if we are taking our time.

How to Do It

Get worked up and in a hurry. Try to rush through the app to complete tasks, or get information as fast as possible.

Simulate different conditions:

- I am about to miss my plane, train or bus
- I need to make a purchase before a certain time, and I have left it to the last minute
- I need to submit time-sensitive information, and I have left it to the last minute

What to Watch For

You will be more impatient and anxious, and may have physical changes that occur as well, so your usage might be different. Your gesturing and inputs won't be as precise, and you will desperately need the app to be fast and reliable.

Slow as a Snail Tour

This was shared with me by reader Alexander Khozya. It's the opposite of the rush tour, and can also cause problems, especially due to timing delays.

How to Do It

Take your time, and work through the app at a painfully slow pace. Imagine an older person who is very wary of technology and reads everything on the screen, then takes a moment or two to decide on what to tap or select.

Alexander says: "Do something in the app, let device alone for couple of seconds or minutes or even let it go to auto-lock, then unlock and continue. After continuing/unlocking weird things can happen."

What to Watch For

If there are delays in user interaction that are slower than the people who developed the app typically consider, there can be errors or strange behavior in an app. If the app requires a user to log in to a server via the web, long delays may cause their account to get logged off prematurely. If interactions or activities occur more slowly than the system expects, there can be errors in either the app or the server system. This can be especially brutal with web sites or web apps if they get out of sync with each other.

Tracy's Thoughts

Tracy: As a new tester, (or when testing a new app) the tours were an easy way to get familiar with all aspects the app and test for bugs at the same time. Tours are a quick way to find bugs. They are also a good way to cover all your tracks with easy mnemonic devices. If you run out of test ideas or worry you haven't covered enough, you can work through a few quick tours to supplement what you were doing. Being systematic in tours and acting out different users such as thinking of how grandma would test was great.

I like the gesture frenzy the best. When using a mobile app, sometimes your fingers don't go exactly where you want, or D> it's in your pocket and things get swiped or pushed by accident, so the gesture frenzy actually simulates actions that the programmer may not have anticipated, just like a real user would use it. Shaking the device, and switching from landscape/portrait produced a fair amount of "real-life" movement bugs that others had missed.

Be creative with tours, and be sure to make up your own, like the toe-swipe!

Final Thoughts on Tours

Testing tours seem to work especially for mobile apps because they can help us rapidly change perspectives and find out new information. Also, since we use them on the move, and need to interact without the devices physically, tours are a natural fit. They can be used for tools of discovery, for test idea generation, or to help focus test efforts in high value areas that we want to explore in detail.

Chapter 5: Test Using Different Perspectives

"It is a narrow mind which cannot look at a subject from various points of view."
-George Eliot, Middlemarch: A Study of Provincial Life

Incorporate Combinations into Testing

Mobile devices are designed for combined activities. In fact, we usually them while we are doing something else. They pop up in our lives while we are on the move, watching TV or sitting at our desks, in restaurants, shopping … pretty much anywhere we can go, we have our electronic leash with us. Mobile devices also depend on a combination of technologies and events to provide us with the things we need from them. When you test, it's perfectly fine to isolate features and affordances, but you get a lot of results quickly if you combine things.

If you use a device in different locations, think about aspects that each context has on the device and the app you are testing. What comes into play when you are walking,

on a bright sunny day and check your email? There are several technologies at work to support that.

When you are working on the device itself, what else is the device doing? Are there other programs running? Is the device trying to find a better network connection? Is it downloading an update to another app? What happens if you listen to music at the same time? If you haven't noticed, or haven't tried, you'll find out a lot about how your app works when it has to compete for resources.

Does the device interact with hardware such as a microphone, speakers or the camera? If you take a picture and use the app, what happens? What about recording a short note for yourself with a dictation app?

Finally, what about combining what you are doing and your motivations and fears into your testing? How easy is the app to use if you feel anxious, rushed and in a hurry? Is it as easy to use as when you are feeling relaxed and comfortable?

All of these are quick and simple ways that we can combine what we do, or exploit the combinations of technology the devices support to find important problems quickly.

Change Your Testing Perspectives with I SLICED UP FUN!

When I made the transition to smartphone testing, I was asked to lead efforts for a new application. It wasn't very re-

liable, in spite of a lot of testing effort. When I investigated, the testers had just copied their test plans and test cases from web application testing and made some adjustments for the differences with the mobile application.

There was a lot of functional testing going on, but the bugs that were being reported by sales people, beta testers and end users were quite different. Testers would work hard, the release would go out, and everyone would hope for the best. As reports came in from the field, the team would scramble to try to repeat and fix each one. This isn't very effective when you are developing a new product, and the Quality Assurance manager was looking to me to help us turn things around. Could we be more proactive in our testing efforts and reduce all these field bugs?

The first thing I did was pull out a testing mnemonic to use in our thinking and to help us expand beyond functional testing. "San Francisco Depot (SFDPOT)" is a simple and effective thinking tool, created by James Bach. Instead of just thinking of functional tests, we can explore the structure of the product, data, platforms, operations and time.

This was a great start, but as I analyzed the mobile devices, the technology they depended on, the communications systems they required, the motivations and fears of our end users, as well as *where* and *how* they used the devices, I had to adjust. I ended up creating my own testing mnemonic for mobile devices called I SLICED UP FUN![7]

[7] http://www.kohl.ca/articles/ISLICEDUPFUN.pdf

(Actually, I just had all the letters. My friend Jared Quinert, an expert tester, came up with the this fantastic combination of my jumble. Thanks Jared!)

I SLICED UP FUN! is a simple thinking model for testing mobile applications. Each letter stands for an approach or consideration when testing mobile apps. You can use it to help your brain change your testing perspectives to find important problems quickly.

While this may seem overly simple, or a bit silly at first, it is an incredibly powerful thinking tool. You can expand it out for as much time as you need, or you can compress it into a short period of incredibly productive time. It's great when we have a lot of time, but on mobile projects we are often under extreme time pressures. Mnemonics like this help you quickly organize your thoughts, and take action. You can either use them to get right down to testing, or to help guide your strategy and planning efforts.

Imagine that you have an hour to test an app. Most people will work through some familiar patterns, and may get caught up in one particular area. Now imagine slicing that hour of time up and devoting a few minutes to each of these perspectives. You might find one or two that don't fit, so you skip on to the next one. In almost every case I have ever witnessed in my career, the person who uses a thinking tool to change their perspectives will find far more important problems than the person who relies on luck.

Mental Checklists

My friend David McFadzean describes mnemonics as "mental checklists." I SLICED UP FUN! is just one of several that I use. Medical doctors, airline pilots, and others who are in critical situations use these tools because they need to think quickly, and thoroughly. Mnemonics are memory aids, so they are made up of words that are easy to remember. Some well known examples are "ABC" (check patient's airway, breathing and circulation) in emergency first aid, "sohcatoa" (soak-a- toe-ah), used to calculate angles in geometry, and "Every Good Boy Deserves Fudge" and "face", used to remember the notes in written music (on the treble clef).

Now let's make this mnemonic do more for us, and combine things when we test. Remember earlier I mentioned combined activities, combined technologies and combined emotions and motivations? The *time* bit in James' mnemonic is the key here to think about. What is happening at the same time during some sort of event? What's going on inside the device? What's going on out there in the systems and networks? What is the user doing at that exact moment, and what are their motivations and fears? Take a snapshot at any given time, and analyze those combinations. It's fascinating, and the heart of excellent testing and debugging.

Time. Timing. Good times, bad times. Think about it as you try out some of these testing combinations.

Now grab your device, follow along to see which of these applies to the app you are testing, and let's slice up some fun! (Don't forget to refer to previous chapters in the book if you need to brush up on how some of these features work.)

I: Inputs into the Device

These represent the ways you can interact and control the device. You can do this with your fingers, your voice, by moving the device around, using accessories such as a stylus or special gloves, or keyboards and pointing devices. there are lots of options.

Test various inputs: typing (tapping) entries while stationary and while moving, and on some devices, using different keyboards. Combine gestures and typing with movement so that sensors, location services and wireless optimization technology are working at the same time. Try entering in data on each required screen in portrait, then in landscape. Do both views work? On smartphones, try entering data and navigating with one thumb, which is a dominant trait for people using the devices.

Try a gesture tour on each screen: input each gesture type: tap, press, press and drag, press and rotate, pinch, spread, double tap, double press, swipe, flick, shake. Web sights or web apps sometimes handle unexpected gestures strangely,

and web browsers reserve some gestures for themselves, so there might be collisions between the app and the browser.

In particular on web sites or apps, count the taps, gestures and other user inputs. Are there too many? Are the gestures and required inputs time consuming or difficult to navigate?

Example Inputs

Gestures

- Touch screen gestures (tap, press, drag, pinch, expand, etc.)
- Movement

Typing

- Explore the built-in keyboard/keypad options
- Utilize different keyboard options if your device supports them - autocorrect, different input or keyboard types
- Different characters - use symbols, chars from other languages, etc

Sensors

- Touch: (gestures and typing)

- Movement: accelerometer, gyroscope, magnetometer
- Light: (dim or brighten screen)
- Camera

Voice

- Use voice control to launch applications or enter information into the device
- Use natural language interface tools to manage the device, load applications and work through the app you are testing

Others

- Use the camera to utilize real-time images, photos, video
- Sync with other devices
- Power cables for charging (inputting electricity)
- Peripherals that you can plug in to the device
- Controlling another device with your mobile device, such as a television or another computer as a remote control

Example Test Combinations

Maximize the different functions of the device at one time.

- Hold the device differently when typing or gesturing.
- Move the device around when you are typing or gesturing.
- Rotate to force orientation changes when inputting anything (using your voice, plugging in a cable, typing, gestures, etc.)
- Move the device when recording voice, or using voice commands.
- Try one-handed, or both hands when using the device.
- Use your non-dominant hand. If you are right handed, try the app with your left, if you are left handed, try your right hand.

Test Tips

Test the application and try out all the inputs that you can think of. If it is a touch device, make sure you try different combinations with your fingers. Keep a thumb on the edge of the screen and try to type, or tap your fingers on the device while using it. If it rotates to portrait and landscape modes, make sure you try in different modes. See if the application behaves strangely while you rotate it and try to manipulate with touch gestures or typing on a keyboard at the same time.

Even the way devices are held can impact their connectivity and how well they respond to application inputs. Can you hold the device in such a way and interact with it so that it causes the app you are testing to crash or

freeze up? What happens if you launch the application in each supported orientation of the device? (Sideways, upside down, straight up, etc.)

Don't forget to record a dictation, direct the device with voice control, or get creative and sing into the device.

Bug Story

One of the most difficult intermittent bugs that I had to track down in a mission-critical application had to do with a freeze up, or gimbal lock. The devices would freeze up, (requiring a hard reboot) during sales presentations, product demonstrations and beta tests. This was unacceptable, because how do you sell an app when it behaves like this? One of the testers found that they could get the device to freeze up if they were performing gestures on the device with one hand and accidentally pressed the screen with their other hand at the same time.

The programmers fixed this issue, but it kept cropping up. I asked the product manager to walk me through an exact re-enactment of what happened right before the last device froze up during a demonstration. It took some convincing, but I had them stand up front, and I sat at a table. WHen it came time for the interactive portion of the demo, he handed me a device. I noticed something. He was touching it (which sets of sensors and gestures,) he swung it in an arc towards me (setting off more sensors,) and causing the screen to change orientation at least once, then for a brief second, both of us were touching the screen. Aha! Lots of

things are being input at once, and since this app depended heavily on gestures, I suspected it was getting overloaded with inputs.

I emulated this situation by moving the phone using an arm curl motion, cupping the device so that my thumb was constantly pressing it, while tapping and gesturing with my free hand. Presto! A so-called "unrepeatable bug" was repeated with ease, and we were able to eradicate it from the system.

S: Store Submissions

For many platforms the only way your customers can install and use your application on their smartphone or tablet is to download your app from an application store. If you want people to use your app, they have to download it from an application store.

We're used to being in control of our own distribution using the web, but many mobile platforms are more controlled: you have to go through a store submission application, see if your app gets approved or not, and if so, the store will make it publicly available. No store, no app to your users.

(Note: most providers have enterprise or business accounts that allow you to distribute internal apps within your organization without using the store.)

Each platform for smartphones and tablets has their own development guidelines, and their own store submission guidelines. Review them early and incorporate them into

your testing. If you don't, you may not get approved, causing delays and costs to your project.

Different stores have different guidelines. Some stores like Google Play for Android apps are more permissive. Others like Apple and Microsoft have stricter guidelines, particularly with regard to development guidelines for the user interface. Be sure to download, study and understand both the development guidelines and store submission guidelines for every platform and store your app will be distributed on. Be sure to look for areas to test to help mitigate against app store denial.

Examples

Store submission guidelines provide information on what they are looking for when they decide whether to list your app or not. This can include:

- App design and look and feel
- App stability and error handling
- How location services may be implemented
- That you ask permission to use private data, such as the user's home address or current location
- Apps they feel to be objectionable or illegal

Test Tips

- Incorporate store submission guidelines as tests early on - they are just more models of coverage

- Use store and dev guidelines to help create new models of coverage, and to help you generate more test ideas
- Re-test often
- Stay up to date, the rules change!

Bug Story

The first store submission I was a part of got rejected for what seemed like a strange reason. While we followed the dev and store guidelines, we didn't read closely enough. Our app passed in all areas but one. We used a non-public library in the development framework API. Just because we could find it and implement it didn't mean we could use it. Instead, we were supposed to use a similar library that was actually written about in their development guidelines. It was a simple fix, but required several days of regression testing, and put us behind on other projects by about two weeks.

This is difficult to test in an app on a device, but make sure you go over the submission guidelines with programmers so that they can determine whether they are using non-standard API calls that could result in a store submission rejection.

L: Location Services

There are several ways that an app can determine our location, so don't assume it is just using GPS, or a wifi

router location. Many devices use a blend of technologies, such as assisted GPS, which uses cell towers to help determine location as well as Global Positioning Technology (processing, wireless and satellites.) Location may also be determined using wifi router locations, or cell tower locations. It can also incorporate movement sensors to get even more accurate information. If possible, talk to the developers and designers to find out specifically what technology they are using for location services. Remember to combine activities - if the device just has to work on location without anything else going on, it can be relatively straight forward.

Location is vital for ease of use and to help determine where users are using our apps so that we can fill in information, or save them from extra inputs or work. We can use them to determine that specialized content relevant to where they are right now is presented to them. Wrong location information can be a huge problem. If there is poor weather, cloud cover or tall buildings, that can interfere with accurate GPS. Electrical activity around us or in the atmosphere, interference from other devices or objects and steel in structures such as tall buildings, door frames, or reinforced concrete can interfere with cellular wireless. Network saturation due to many people using cellular services from the same area can cause problems with location determination.

Real world testing is vital, especially in different kinds of locations that have varying wireless services and sig-

nal strengths. Different geographical locations may have unique conditions that make them difficult to pinpoint. Location services require user permission on native or hybrid/installable apps, so it is also important to test what happens if the user refuses permission.

Examples

Services

- Global Positioning
- Using Cellular Tower Locations
- Web IP address location

Sensors

- Accelerometer
- Gyroscope
- Magnetometer

Others

- Device id

Example Test Combinations

- Incorporate movement and sensors. Walk briskly and move the device around.

- Start and stop many times, which can throw off location algorithms that calculate where you will arrive next based on your movement speed.
- Explore network connection issues:
  - Move in and out of dead spots
  - Move from one network to another (eg. Wifi to wifi, wifi to cellular, cellular to wifi, EDGE to 4G, etc.)
- Try it indoors.
- Try it outdoors. (How does the app handle transitioning between indoors and outdoors, and back again?)

Test Tips

What effect might walking, running or traveling in a vehicle have on location accuracy? How about poor weather or other kinds of interference? Steel can interfere with wireless waves, so also try it near or around tall buildings. Also try it at different times of the day when cellular towers might be congested.

Bug Story

I have an exercise app that tracks trips when you are walking or running. It tracks your trip and displays it on a map, as well as how long it took, how far you travelled and things like that. Some days, the trip information is bizarre. The line that shows my trip is usually very straight with tight angles as I walk on pedestrian paths in a park

and on sidewalks downtown. Sometimes, it is a jagged line jumping back and forth sometimes half a block in either direction. It also calculates these bizarre jumps in the trip distance. The output is completely bizarre and useless. I tend to get this result at the end of the day during rush hour when cellular towers are often at their busiest, and near tall buildings when I am in the downtown portion of my trip. It also seems to happen more frequently on overcast days. I am also starting and stopping more frequently due to heavier traffic.

I: Interruptions and Interactions

I was researching different test ideas for mobile devices a few years ago, and I stumbled across this great blog by TestLabs. It is sadly gone now, because it was a really good reference for smartphone testing when they were just getting popular. One of the ideas I got from their blog was to test how an app deals with interruptions and interactions. It seems on smartphones and tablets, the answer for many apps is: "poorly."

It is difficult for mobile apps (native, web or hybrid) to handle interruptions from error messages, low battery, loss of signal, and other kinds of messages. Calendar event reminders, system notifications, and messages from other apps that interrupt current usage are important to incorporate. Context switching between apps, such as leaving a web page or app when it is loading slowly to do something else, and then returning can cause apps to end up in a

strange state. Even something as simple as momentarily obscuring or covering up your app screen with a dashboard or other informational screen can cause problems, especially for web apps. Many will log a user out of their current session when interrupted, or if the web browser is sent to the background and brought back up.

Interactions with other apps can be problematic, especially if the file sizes for downloads in the app (or especially a web site web app) are large, and if there is a lot of JavaScript or other client side processing. If the device is working hard, and there are a lot of apps running in the background, it can cause performance problems or timeouts in a client/server situation.

If you have a lot of apps running on a device, they may starve the resources your app requires for optimal performance. Some apps interrupt a lot to notify you of changes or information you need to be aware of, and they might not interact well with your application.

Example Test Combinations

- Run multiple applications at the same time
- Switch between apps, return to the one you are testing
- Use another application, then using the application you are testing (try built-on apps like email, calendar, texting, note taking, others)
- Create conditions to create notifications to interrupt

what you are doing (new emails, phone calls, text messages, other notifications from other services)
- Create error messages (losing network connections, notifications, low battery, plugging in a power or other cable, others)
- Determine how well the app recovers from interruptions. Can you pick up where you left off?

Test Tips

When you manage to get your device into different types of states that cause interruptions, you will probably find other interesting things. For example, when we started testing interruptions from a low battery warning message, we found out that the device itself behaved differently on a low battery. We also found that if we were running several apps at the same time utilizing multi-tasking, the app performed poorly and we had to go and shut some down. One app I tested would warn you of this condition, and would ask you to shut down apps and start again. Most don't, so you aren't sure why it isn't working as well as it did last time.

Bug Story

A fascinating bug occurred with one app. If we used the calendar app to create or edit an event, recorded a voice dictation or a note, added a new contact, took a photo or video, or added a song to our music library, then launched our app, it would always fail to connect to the internet.

There would be an obscure, gibberish-looking error message, and then the app would crash. Once you restarted it, it was fine. Do you notice a pattern in all those actions? They save something to the device. So for some reason, if we saved something on the device, then started the app, the network connection would fail. It wasn't our fault, it was some low-level error in the device, but we had to put in error handling to work around it so that our end-user's wouldn't be impacted negatively.

C: Communication Integration

A core underpinning of mobile devices are various communication services, whether they are texting, video chat, SMS or voice (telephone or VOIP.) How does an app handle integrating with communication? Think of all the ways and stages in your app where a communication interruption could occur, or moving away from your app to communicate with someone, and then returning.

Communication integration can be awkward. It is common for people to use their cell phone while they are using your application or visiting your web site or web app. Some apps provide a support line to call if there are problems to get assistance, but how does that integration work? An app may have a phone number for you to tap, or utilizes a special gesture to make a call. Does the app survive that interaction once you return to it after the call, or does it log the user out, or crash? Some apps also have integration with phone calls or video chat built-in as part of their normal

workflows and usage, so those paths need to be tested as well.

Interruptions from communication can be incredibly awkward for apps to handle, and they can be numerous. SMS texts, instant messaging, email, video chat and other programs are constant interruptions during the day. Cell phone calls are frequent, as are notifications about missed calls, voicemail and other events that we have missed.

Examples

- Interruptions: you are using the app, and someone tries to communicate with you using any number of methods
- Integration: while you're using the app, you need to communicate with someone, to ask a question, or inform them of something you are viewing in the app.
- Roaming: does the app work properly if you are in a different location, using the roaming service? If it does work, will it cause unexpected charges for the user?

Example Test Combinations

- Take a call while you are using the app, and go back to it
- Ignore or blow off a call
- Listen to voicemail and return

- Ignore the voicemail notification
- Copy text from the app, and paste it in an SMS message
- Take a screen capture of the app and email it to a colleague
- Work through the app, stop, phone a colleague with a question on how to proceed, and then go back and use the app

Test Tips

Think of scenarios where communication can either interrupt or enhance your app experience. These scenarios can get quite complicated, so start simply with a decision or classification tree. It's amazing how simple communication interruptions in a workflow may not have a negative impact on an app, but more involved, yet similar scenarios can fail.

Bug Story

I've tested several smartphone apps that would freeze or crash when interrupted by a phone call, voice mail alert or text message.

E: Ergonomic Considerations

Smartphones and tablets are smaller than PCs, and we don't use them in ideal conditions. A PC at a desk can

be placed at just the right height and angle, and we have special chairs, keyboards, monitors, mice and other devices to help us stay comfortable and healthy. Mobile devices have to be held, or placed on a flat surface. By their very design, they aren't that helpful when it comes to comfortable use. They put us under more stress. App designers and developers need to take this into account, and do what they can to make apps more usable and reduce our stresses wherever possible. If they don't, people will get sore, angry and stop using the app.

Watch for anything that causes strain while testing. Do your eyes get sore quickly? Do you get frustrated and want to stop using the app? Do your fingers get sore, or do other muscles get sore when you use the app while moving around? These are all clues that there are usability issues. Count gestures, count taps, count the amount of fields that need to have information entered and count the screens that people need to go through to complete tasks or reach goals. Suggest areas to avoid textual input where possible, with alternatives such as checkboxes, radio buttons, drop-down pickers, location services, etc.

Watch your emotions. If you feel angry, frustrated or want to avoid the app, it is likely causing physical discomfort. If you notice that you are feeling physical discomfort or pain, research more deeply to find the source. That will be a usability problem, especially if you use the app standing up, walking, etc. Apps are harder to see and interact with while on the move.

An app causing physical pain can be difficult to notice at first, because we don't think to observe it. If you find yourself avoiding working with the app, get a friend to use it and then watch them closely. Do you see signs of physical pain in their eyes or facial expressions? How about with their posture or with their hands? Do they stop and rub their eyes, or their shoulder or wrists? These are all telltale signs of something causing pain which we may not notice in ourselves, but are obvious when we watch others.

Examples

We get sore fingertips, sore arms, sore backs, and sore eyes. This can be caused by workflows that demand too much typing or scrolling and gesturing. The size of text, controls and touch targets may be too small. Colors may be harsh or bright, or difficult to see in different lights.

- Sore fingertips, forearms - does the app require too much typing or gesturing?
- Sore eyes - are items too numerous or too small? Are colors too bright or dim (causing a squint reflex), or are they difficult to see in bright or dim conditions?
- Sore back - does the app influence you to hold the device awkwardly?
- Watch closely how screens load or how the application redraws itself when you change the orientation. Brief flickers can be really straining. If there are a lot of them, or other slight movements that are rapid and numerous, they can make people feel sick.

Test Tips

Watch for usability problems in workflows or app functionality that makes strain worse. If you feel frustrated or even angry when using an app, stop. Take a break. Pay attention to your body. Does anything hurt? Do you feel strain? If so, investigate to see if the app is making that worse.

Bug Story

One morning, I came into work with my coffee made just the way I like it. It was sunny, and there was a cool breeze. The testing team was in a good mood, and there was a lot of joking around and witty banter. There were three of us working full time on smartphones and tablets, and the rest of the team were working on PCs and web apps. An hour later, we had a daily standup meeting, where we talk about our progress and any issues we are having as we stand in a circle. Everything was fine and dandy, but the mobile testers were now grumpy. Angry even. They couldn't tell me why. I started to watch them, and they were rubbing their eyes, and rubbing their necks.

I observed them further, and noticed that they were leaning forward and squinting at the screen. The others were sighing, muttering under their breath and threatening to throw the mobile devices out the second floor window. I decided to investigate further. Whenever anyone was leaning forward and squinting, they were working in the main area of the app. It had a black background, with

lighter text and control colors. (It's often the other way around.)

When they got frustrated, I stopped them and asked what was going on. It turned out that two changes to the app were causing pain. A slight color change had been introduced, which made the text a bit darker than it had been before. That was enough to cause people to have trouble seeing the app properly. Another was due to more work caused by a login screen. The app no longer allowed autocomplete of user names, and the security group in the company demanded that passwords be much longer with a mix of different characters. Once we figured out what was wrong, the designers and architects of various groups worked together to use our information to make the app more usable.

Another issue is quite common with cross-platform mobile development frameworks. They aren't native apps, so they require more processing and rendering. You'll notice jumpiness or screen flickers from time to time. This can happen when new screens load, or when a device is rotated and the orientation of the app changes, or whenever a user interacts with the app.

Once, when I was working with Tracy Lewis, I saw her wrinkle up her face as she was changing the orientation on an app. When I had tested the app myself, I thought the screen was flickering, but I decided it wasn't that bad, so I didn't note it. I did notice that I didn't want to use that app very much, so I wanted to explore more. After I saw

Tracy's reaction, I took my device, held it up to her face and rotated it back and forth a few times. She pulled back and covered her eyes. Now we had a serious bug to log - the app was training users not to use the app, with a kind of Pavlovian response. The technology was causing physical strain.

Some cartoons have been known to cause seizures because of bright lights and lots of flashes. A jittery screen redraw on a device can have a similar effect. If it happens a lot in an app, it can make some people sick.

D: Data - Inputs and Outputs

I borrowed this from Bach's San Francisco Depot test mnemonic. Data is anything that we feed into an app, and what it spits out. That could be as simple as typing in symbols or letters to enter in a user name and password, or the app saving a note or other file. Many apps use databases, which are programs that manage data efficiently for us.

Try different text inputs, with different character types from other languages. Try classic input validation tests: input overflows, invalid data, SQL or other injection, or no data input and see what happens with error handling. Determine if there are any data feeds or dependencies on other systems, and test what happens if there are no connections to web services or other sources of data via the network.

Examples

- Anything you can type and enter into an application.
- Application storage: files, databases, settings
- Outside sources of information: data feeds and application updates
- Updates to software and firmware (you download files, and they get updated by your system)

Example Test Combinations

- Test on the move. Try data-related actions while moving the device (get the sensors going as well.)
- Utilize different network types and transitioning between networks.
- Have the app process different sizes of files, and repeat your actions several times.

Test Tips

You can often quickly enter in different types of characters by holding down a vowel on an English keyboard. Other options for other languages will appear. You can also get different options for punctuation and symbols by using a press instead of a tap. Many apps stop functioning as soon as you enter in non-English characters.

Explore options to generate and input different sizes and types of data. Testing consultant Elisabeth Hendrickson

recommends using the Goldilocks heuristic[8]. Try data sizes that are too large, too small, and just right. Record large video or sound files, or try to download a file that is very large and see what happens. Mobile devices have more stringent storage restrictions due to the devices being smaller and having less disk space than a PC.

Another thing to try while entering data and you see the processing wheel, repeat the of special characters. Sometimes they can only handle one of each, or can only deal with searches or lookups of smaller entries.

Also get sensors and other apps engaged while entering text or saving a file.

Repeat data entry or processing over and over to see if you stress out storage and processing, or overflow displays. Add time and timing of inputs to add variability.

Add in different network types, speed and transitions with different particularly large sized files.

Bug Story

Here's an easy one. While I was working with mobile tester Tracy Lewis on an app testing project, she decided to repeat entry of the Pounds Sterling symbol. On the device she was using, if she pressed the dollar sign, the pound symbol appeared as an input option. She entered several in a row, and the app went into an error state. The app would not

[8] http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf

work at all anymore unless you went to the home screen and started over.

I've seen countless errors with different kinds of symbols, with files that were too large, or were of the wrong type, when the program expected something else.

U: Usability

Usability is the single most important factor for a mobile app. It is so crucial, I recommend that you study mobile usability works so that you can find and report important usability problems quickly. As I've mentioned before, if your app isn't very usable, people will just delete it. (The *deletable offence*.) If it annoys them, they may just go on to a social networking service and rant about your app. (A *rantable offence*.) Or, they may just quietly give your app a poor rating on application stores.

If a potential customer searches for your app, they may go to a competitor's app because they have a higher rating. Think about it for a moment. When you search for an app to satisfy a need, and you find 3 apps that fit the bill, but one has a 1 star rating, another has a 3 star, and another has a 4 star. Which one would you choose, all else being equal? I even might pay a couple dollars more for the 4 star. A poor release might yield your app a whole string of 1 star ratings, which is incredibly difficult to pull back from. Most people won't say when something is good, but if they don't like it, they will complain in some form or another. It might take several stellar releases just to get that rating back up

to a 3 or a 4. In the meantime, you are potentially losing customers to a competitor's problem.

If you have worked on professional software teams in the past, you are probably used to usability bugs getting the lowest priority. Technical teams get upset about problems like crashes, data deletions, or something else that is potentially destructive. Non-technical people tend to have the opposite view. They tolerate crashes and deletions (often blaming themselves), but they get very upset if an app is hard to use. In our quick fix society, we also don't have much patience. If you are used to usability bugs getting ignored or deferred, it is time to flip them over in order of importance. You need to address the issues that are important to your users first, and then start working on other important issues like app crashes and corruption.

Getting Usability Bugs Fixed

To increase the chances of getting usability bugs fixed, cite examples from experts in your bug reports. Here are some I recommend studying: Luke Wroblewski (author of *Mobile First*), Josh Clark (Author of *Tapworthy: Designing Great Mobile Apps*), Brad Frost (responsive design and mobile-first proponent), Alan Cooper (user experience expert) and Steve Krug (author of *Don't Make Me Think*.) Smashing Magazine[a] is a fabulous online resource that I constantly

refer to when I test and log issues.

[8] http://www.smashingmagazine.com/

Usability and human/computer interaction is a massive topic. You can spend an entire career studying and implementing usability testing, or as a User Experience (UX) designer for example. However, you don't need to be an expert to do it well. Draw from expert work, and listen to your own emotions and impressions. I like usability expert Steve Krug's definition:

> After all, usability really just means that making sure that something works well: that a person of average (or even below average) ability and experience can use the thing - whether it's a Web site, a fighter jet, or a revolving door - for its intended purpose without getting hopelessly frustrated. - Steve Krug, Don't Make Me Think, 2000, p. 5

Krug wrote an interesting book called *Don't Make Me Think* that describes web app usability issues and how to address them. He has a handful of clever insights that have stuck with me as I test apps, particularly mobile apps:

Don't Make Me Think

Apps Should Explain Themselves

Don't Waste My Time

We Are Creatures of Habit

Make it Easy to Go Home

Let's briefly explore these insights. Krug makes a great point: we don't want to stop and concentrate and work hard to try to use an app. We depend on our devices to satisfy our motivations, whether that is for problem solving, entertainment, communication, or to merely satisfy curiosity. We don't want to interrupt our flow of thought to meet our needs and struggle with a device or application. We just want to do whatever it is we want to do. The device is the medium, not the thing in itself. We don't want to think about things and solve unrelated problems first. Therefore, an app should explain itself - we shouldn't have to read a manual to use it. We should be able to open it up, and it should be obvious on what to do next. An app should do what it says it does, and it should be simple to perform actions.

The ultimate waste of time is an app that doesn't match what it is advertised to do, but other apps may be useful, but cumbersome, needlessly adding precious seconds or minutes to our activity due to their poor design. Once we figure out how to use an app, we tend to use it the same way ever after. We also project that experience on other, similar apps. Usability experts try to identify common patterns of use, and cater their app designs to follow these

patterns. They observe many different users, research, and compare notes to help determine optimal designs. These are still emerging in the smartphone/tablet space, but we're starting to see some good points that we can use when testing. Finally, we make mistakes or change our minds, so an app needs to give us an out, or a way back home so we can start over and do what we really meant to do. If we get caught in an app and have no way to get back to correct our mistake, it can be incredibly frustrating.

Examples

Luke Wroblewski wrote a fabulous book called *Mobile First*, which was published in 2011. He coined the term *mobile first* and has provided guidance on how to make apps that are more mobile-friendly. Here are some items to explore and evaluate the app you are testing from this book:

- Simplicity! Apps need to be simple so that they are prioritized for what really matters. (p.28) If an app demands too much typing and gesturing, look for ways to trim it down.
- Location and time - people use devices differently than PCs, and in different situations, often for short bursts of time. (ch. 2) Use a tour and test in short bursts. Mix it up by trying out different emotions for different bursts.
- Location detection: saves time and allows for information or content to change depending on where

you are. (p.36) If an app forces you to enter a location, or to find location-dependent information instead of doing it for you, then there is a problem.

- Relevant and well-placed navigation options make apps easier to use (p. 66) It's easy to get lost in an app, so the navigation options (especially a way to go back, or go home) need to be obvious, easy to use, and consistent from screen to screen.
- Go big with touch targets - they have to be easy to interact with. (p. 84) If you have trouble figuring out what you can interact with, or you can't interact with the items on the screen at all, or cause side effects by accidentally interacting with something else, you'll have problems.

Test Tips

Read Apple's *iOS Human Interface Guidelines* and Microsoft's *User Experience Design Guidelines for Window's Phone* and *Android Design* guidelines. If you are working on apps for those platforms, the app will need to conform to their guidelines for it to be certified and allowed on their application stores. If you aren't developing for these platforms, you will still learn about great usability from two of the leaders in computing. You can apply the lessons they teach on any modern device.

Focus on first impressions. Is it easy to install the app and get going, or is it cumbersome and difficult? It's easy to forget the first time after you get used to the app, so make

sure you pay close attention when you first start testing it. If it is too familiar, get someone else who is unfamiliar to find, install and start using the app. Observe and note anything they struggle with.

Become a student of mobile usability. Check out the work of people I have mentioned in this section. For easy to digest content and summaries, Smashing Magazine has fabulous a summary of mobile usability issues: The Elements Of The Mobile User Experience[9]

Test the Visual Design:

- Make sure the layout of the app is clear and clean, there aren't too many items cluttering up the view and confusing the user, items are large enough to be seen
- Can you see the elements on each screen clearly while you are moving, especially while walking?
- Can you see the application clearly in different lighting conditions – bright sunlight, overcast, in the dark, with indoor light, etc.?

Test Various Inputs:

- Are inputs too numerous? Does it take too long to complete a workflow because there are too many steps?

[9] http://mobile.smashingmagazine.com/2012/07/12/elements-mobile-user-experience/

- Are there side effects when you try to interact with the app, something unexpected occurs due to a "fat finger" gestures or inadvertent gestures? This usually means there are too many items, or items are too small, or the touch targets on objects are too small.
- Can you interact with the application with gestures, voice control, etc. and enter in data and complete workflows while moving (especially while walking)?
- Are error messages clear and helpful so that the end user can complete tasks or workaround if conditions aren't ideal? Do they provide information to help you solve a problem and get something done, even if the issue is unrelated to the app?
- Is there visible feedback to let a user know that their action was recognized or is being processed?

Test App Workflows:

- Are the steps to complete tasks or work in the app intuitive and simple?
- Can you complete tasks quickly and smoothly?
- Are there too many screens, which causes productivity or app enjoyment to suffer?
- Are there too few screens that are trying to do too much, with lots of information which cause confusion on what to do or what is going on??

Bug Story

Just last night, my friend Aaron West was over. Aaron is a talented programmer, and we have worked on several projects together. After a while, we realized we both have the same sarcastic, zany sense of humour, and we became friends. We were talking about mobile app design (because that's what nerdies do over drinks) and the apps we are currently working on. For fun, I pulled out a couple of apps that I had recently downloaded, and found they were unusable.

The first app was a search service app that helps you find local businesses and services. I handed my phone over to Aaron, and he tapped the icon to launch it. I studied his face. As the splash screen appeared, it displayed a message that was campy. He snorted with derision. "Wait a minute. Is this just a copy of *app X*? (a well known and popular app in that space.) Once the program loaded, he looked confused. "What am I supposed to do now?" He tapped the first option, and was provided with one listing. He tapped an arrow to go back, and tapped another item on the main home screen. He was then presented with a long, busy list of options. "This just looks like a poor copy of *app X* with very few listings and a poor layout. I'd delete it."

The second app was an app that provides a visual search for local services. Aaron tapped the icon to start it. The splash screen that explains what the app does disappeared almost instantly and was replaced with an ad. He found this confusing. "What did I just launch?" When the main screen

loaded, it forced him to enter his location. He picked an entry from a picker, and tapped a button to proceed. He was then asked to narrow his location down even further, with another set of choices. "What the hell? Why can't this thing figure out my location like a mobile app is supposed to?" After he selected his location, a screen appeared briefly, then it redirected to Google Maps. "Stop! What happened? I don't want to be in Google Maps." It then loaded with pins showing local services, but the map was zoomed out, so he had to use pinch and expand gestures to get a good overall view. He was then stuck in the Google Maps app view, with no way to get back to the app. There were no visible navigation options to go back home. "Yeah, sadly I think I'll be deleting this one also."

Exercise

Your homework is to come up with a list of reasons why both of these apps aren't very usable, using credible sources.

P: Platforms

This is another one I borrowed from James Bach's San Francisco Depot mnemonic. It's an important one, especially in the mobile space. The proliferation of devices types, manufacturers, operating systems, screen sizes and resolutions, wireless connections, cellular carriers and the

technology they use, and different web browsers are enormous. Also consider the different spoken languages you may need to support. In North America, we can get away with two or three languages, but in Europe, you have to contend with more than twenty. (You may also need to take programming languages into account, but I'm not going to touch on that here.)

So why should you care about testing on different platforms? Any of those factors can have an impact on the app you are testing. Your app may work just fine on one combination, and crash, perform poorly or look terrible on another. You can only get so far with emulators, and you'll have to go out in the real world, with real devices, using real services to adequately test.

Different devices have different hardware, operating systems, web browsers and network technology. HTML5, and other web technology is completely dependent on wireless connections to work, and there are different technologies and different environmental conditions to take into account when testing.

The basics to consider for a platform when testing an app that requires internet access are:

- Hardware manufacturer
- Device type, model
- Operating System version and sub-version
- Wireless connection with wifi
- Wireless connection using a cellular network

- Cellular carrier

Try the following:

- Test on handsets from different providers for different operating systems (Android, Apple, Microsoft, Blackberry, etc.) Incorporate as many as possible.
- Utilize different screen sizes and resolutions. It is difficult for apps to work well on different sized screens with different resolutions.
- Be sure to incorporate cellular network testing in different environments: inside an office, in an elevator, while moving inside an office. Outside around tall buildings, while moving around, and also try to incorporate using in overcast or stormy weather. Also try around a microwave in a kitchen; it creates interference for wireless communication.

Examples

With device types, we are currently concerned with smartphones and tablets. Most tablets are now using smartphone technology (think larger smartphone with no cellular phone inside), but they have differences. The larger screen size and different screen resolution are important factors. Images, designs and layouts can look vastly different from one platform to another. Some may not fit, or critical aspects of the application may not display on a smaller screen.

Different manufacturers use different kinds of technology, which can also pose challenges. Some use very different combinations of hardware and firmware to support a mobile operating system. Like anything else in life, there are strengths and weaknesses. One device may have much better screen resolution and display characteristics, and another might download and render things from the web much more quickly than others. However, both of those activities might wear down batteries more quickly.

Operating systems from different companies can behave completely differently. Within an OS type, you can also have large differences from version to version. Even subversions can have an effect: ie. version 2.1.2 may work just fine, but a change introduced in 2.1.3 may cause your app to malfunction.

With a web application, there are a huge number of mobile browsers to consider. Some platforms have one or two web browsers, while others may have a dozen.

Wireless connections for network access have their own set of combinations to be aware of. Different wifi devices have different speeds, and some of them get saturated with a lot of users. For example, at home, we rarely have more than 4 devices connected, and we have a high speed internet connection. When I connect to a public wifi beacon, it is never as fast as when I am at home. Also, wifi signal strengths come into play as well.

When it comes to cellular providers, they use different technology to reach a certain standard such as 3G, 4G LTE, etc.

They also may have different usage patterns, which effect their performance. Some carriers have better performance than others. When you travel, the relationships carriers use to provide you with cellular access are complex. Carriers also send updates to their firmware or special software on your devices, so you need to be aware when this happens, and try to find out if your app is impacted.

Also consider encrypted or secure communications vs. insecure. Secured communications always add more overhead so they can impact performance.

If your device uses accessories, make sure to include them as well in your testing, even if it is using the power cable when the battery is low.

Finally, testing for different languages, and in different countries, or *localization testing* is important to consider. Different character sets are handled differently, and if you the app wasn't programmed to support the inputting, processing, rendering and displaying of characters in different languages, it may crash, or not display them.

As mobile tester Johan Hoberg points out, even if a language is supported, the formatting may be off, and it might look strange in the app. Some letters in other languages are longer, and many words or phrases might be longer or shorter than the native language your development team uses.

It can be difficult to predict where an app might be used, particularly if you are developing it in English. Since people

all over the world speak English, they may still try out your app. I was at an iOS developer meetup a few months ago, and one of the attendees told us that for some strange reason, the app he developed here in Calgary, Alberta, Canada has a huge following in Bulgaria. He has had to tune and improve his app for the unique conditions mobile users face in that country, because they bring in the most revenue for him.

Note: just because literal translations are correct does not mean that the language is clear. Local idioms or practices can have an enormous impact on the meaning of a translation. Sometimes translations use outdated language, or are unintentionally offensive. Be sure to get native speakers who understand local practices with the language to test as well.

Test Tips

Test on as many different hardware and device types as you can. Research what you are planning on supporting to help you narrow it down.

Use different wireless connections with different signal strengths. Try different locations with different connection types from different carriers.

Save time by diagramming out your combinations using tree diagrams first to get the most of your testing. Try to use multiple combinations at once. (eg. Hardware, OS version/-sub version, wireless technology, secure communication) If there is a problem, narrow it down by simplifying.

Bug Story

Recently, I tested a mobile version of a popular web application for a potential client. I had casually toured the mobile version on my smartphone, and it had been slimmed down considerably. It looked nice and clean and minimalist, with a singular focus on each screen. As I started using it, it felt a little *too* minimalist. In fact, it was missing core functionality. When I had scheduled time to test, I used a tablet as well. I set both down in front of me, and repeated my actions on both devices. When the tablet version launched, with a larger screen, there was the missing functionality! There was a row of buttons on the bottom on the tablet version that was completely absent from the smartphone version.

As I mentioned before, any of the areas I have talked about have shown differences with one app or another. Take them into account and plan.

F: Functional Properties

If you have a professional testing background, you will be familiar with functional testing. In fact, it absolutely dominates Quality Assurance and test automation activities. Functional testing usually involves using the requirements or specifications as a guide for testing. The requirements may be written down formally in a thick tome, in stages on 3x5 cards, or vaguely discussed using enhancements like squiggly lines on whiteboards, sock puppet actors, or

interpretive dance. A lot of functional testing starts with these guidelines. A tester gets a sense of the requirements, and compares the app with them. If there are differences, they report them.

However, I urge you to go beyond simple verification and validation when it comes to functional testing. One reason is that we will miss the same things other people miss, unless we are lucky. Another reason is it can get pretty boring. How many requirements docs take some of the other perspectives into account in I SLICED UP FUN? When they are well done, they will have thought of some of them, and when they are done poorly, possibly none of them.

Test Tips

Incorporate device settings into functional testing. There are a lot of preferences that can be changed. Explore your device settings, determine what might have an effect on your app, and try out different combinations. Different combinations of settings beyond the default may interact with your app strangely. Here are just some of the things you can take into account when testing:

- Automatically change wifi providers as you move
- Don't prompt for passwords
- Turn location services off
- Set the device to airplane mode (turns off all sending and receiving functionality)

- Automatically poll for new email
- Change or disable alerts for messages and communication
- Enable call forwarding
- Change sounds or disable them
- Turn on accessibility help (different colors, larger fonts, screen readers, etc.)

Also explore any settings the app itself may save on your device. What happens if you disable all of them or change all of them? Are there nonsensical settings that cause the app to not work properly if they are selected?

Analyze what an app does, what problems it solves and all the features it might have. Tours are a great way to discover things about the app that even the designers and developers may not know about it.

Determine what problems the app solves, or how it entertains or is otherwise useful for an end user. Now compare the app with your findings. Are there missing requirements? Now compare it with a similar mobile app. Is it missing key mobile features that competitors are using to create a better mobile experience? These are important issues designers and developers need to know about.

Bug Story

One of my friends told me about an interesting problem he had with his brand new, shiny and awesome smartphone.

His wife bought him a new device that he had been drooling over for his birthday. He got it all set up just the way he liked it, and transferred over all his information and apps. He also installed a couple of new apps that he had discovered while configuring the device. One day he was driving down the road, and felt heat in his trousers front pocket. His smartphone was in that pocket, so as soon as he could he yanked it out of there. It almost burnt his hand, so he tossed it on the passenger seat. By the time he got home, the battery was dead and the device had cooled. He plugged it in, charged the battery, and didn't notice any problems.

Once again, near the end of the day, his phone heated up. He could barely touch it - it was so hot it burned his hands. Once again, he plugged the device in and charged the battery. The phone cooled down and worked properly. He stopped using the device for the rest of the week and decided to spend his Saturday debugging the problem. He started analyzing what had happened. He noticed that the problem occurred when the battery was low. Low battery on devices can cause all sorts of different behavior.

Many apps will even go into a different mode to try to save energy. He also knew that heat is a byproduct of energy consumption, so the device must be working very hard when the battery was low. He also looked at the apps he had installed when the problem started. There were two of them. He uninstalled one, and performed actions that caused the battery to wear down quickly.

Nothing happened, so he re-installed the other app, and ran

the battery down again until he got a warning message. Suddenly, the phone started to slow down and began heating up. He researched that app, and found out that it turns wifi off when it discovers low battery. He researched some of the other apps on his device, and found another one that turned wifi on.

There was the problem - two apps had conflicting functional properties under certain conditions. One would turn on wifi, and the other would turn it off, so in-turn it would get turned on and off, over and over. This stresses the computing features of the phone, particularly CPU and memory usage, which get maxed out. This causes a tremendous amount of energy to be expended, which causes heat.

Mobile developer John Carpenter notes that high end devices with metal or conductive material in the case can heat up easily when the device is under load. Since the device is designed to diffuse the heat due to optimization services, if it is hot to the touch, then an app is usually at fault.

U: User Scenarios

When I started out in software testing, we did a lot of functional testing based on the requirements, as I mentioned in the previous section. I got frustrated when people would find problems after the release that I and my team were missing. Sometimes it was something really obvious, at least to the people who depended on our software. Sadly, it wasn't obvious to us. I felt like my testing efforts weren't

enough, so I read Alan Cooper's book *The Inmates are Running the Asylum.*

Cooper's concept of *personas* really resonated with me, and was instrumental in helping me completely transform my testing effectiveness. Cooper recommended creating models of real users, called personas, and then to model their goals and activity scenarios to help with application design. Focusing on users, understanding how they would use our software, and what problems they required our software to solve for them was a completely different approach than what I was used to. I took Cooper's "goal-directed design" and used it to create test ideas.

With practice and a bit of trial and error, I stopped looking only at the requirements and functional testing, and started with users. I complemented the functional testing with user-based tests. The developers were happy I had double checked how they built the app by comparing with the requirements, and our customers were happy because I managed to discover, log and get many important usage-based problems fixed.

You can get quite involved when you create personas, or you can use something lightweight and simple. In the paper Personas: Practice and Theory[10], Pruitt and Grudin describe personas:

> "Personas consist of fully fleshed out fictional

[10] http://research.microsoft.com/en-us/um/people/jgrudin/publications/personas/designchapter.pdf

> characters, as might be encountered in a film or novel, given specific ages, genders, occupations, hobbies, families, and so on. Photographs and considerable supporting information is provided for each of a handful of Personas used in a project."

I have created personas in this way in the past. I would talk to sales and ask them who our typical customer was. I'd also talk to customer support, and try to find out about different customers. In particular, I wanted to know the problem-customer. Who complained about what and why? If there was a local demo, training course or technical troubleshooting problem, I volunteered or begged to go and see real customers at work with our software. When I could, I even talked to different customers, and asked them questions about what they did, what their hobbies were, and whatever I could get away with. If I talked to ten people, I could probably create three different personas based on composites of different user types.

Personas

Nowadays with privacy legislation and different laws and political correctness, you have to be very careful when you ask people personal questions. It's hard to get specifics to create the full-blown personas, and I have found I can get by with approximations. You can observe and infer through what you see and hear. I also found that creating personas

based on extreme users could be very useful in changing perspectives and finding new problems.

One team I worked on used formal usability testing, where you put your software in front of real users, and give them objectives to complete, without prompting them. We had a User Experience consultant who facilitated the tests, and trained us how to conduct the tests ourselves. After we implemented her design and incorporated the initial usability test feedback, it was time to try it again. We recruited 7 testers and headed out to test. (Our UX consultant told us that even though we had many thousands of users, we could get enough useful information with a sample size of 7.)

I noticed three patterns in our users: one user was older, near retirement. As we sat at her desk, we noticed that she had pictures of her grandchildren. She was a bit of a technophobe and we made her nervous. She kept apologizing for being slow and "not understanding computers that well." We got her to relax by talking to her about her grandchildren, and she got through the usability testing, only struggling in a couple of spots.

The other users were quite similar, save one. Most were reasonably tech savvy, and unlike the older user, did not read any directions at all. The final user was a very attractive, young woman. The young, single, male programmers were quite eager to facilitate usability tests with her.

It was amusing to watch them try to impress her. However, they were quite shocked when the pleasant outward

appearance did not match the unpleasant experience they had working with her. She was angry, and yelled at them about the design changes that she found slow, or difficult to use. They went from looking foolish as they tripped over each other to work with her, to tripping over each other to get away.

I was an observer, and I noticed that this experience was incredibly powerful. The juxtaposition of an attractive person with an unattractive, aggressive personality was surprising, and stuck with the team for the remainder of the project.

I created three personas based on what I observed:

1. George: An older technophobe who is slower, but more careful and methodical. Key emotions: fearful until they solve a problem, then a sense of happiness and pride. Key app characteristics: ease of use, and providing lots of help.
2. Joanne: A middle-aged user who is comfortable with technology but tends to rush. Key emotions: flat and indifferent. Key app characteristics: speed and ease of use.
3. Mary: A younger twenty-something user who hates anything that gets in their way. Key emotions: anger and hostility lurk just beneath the surface. Key app characteristics: extreme ease of use and fast performance.

Instead of using information from the people we observed, I created personas based on our coworkers, parents and grandparents. We created names for each one, and imagined their motivations and emotions while they were using the software. It might sound a bit silly, but we all got really good at pretending to be one of those three people. If there was a contentious usability issue, we just had to invoke the memory of the person that the Mary persona was loosely based on, and the developers who had been at the usability test session would scramble to fix it.

It can be surprising to find out how people who are similar to you, who have similar likes and dislikes and experiences may use software very differently. Reviewer Elizabeth Lam points out that people who learn to speak other languages, such as some Asian languages may read and write from right to left, instead of from left to right the way English speakers do. If you have a lot of people who are using your app whose native tongue isn't English, they may look at the layout and design differently, and may interact with it differently. For example, think about gestures: instead of trying to swipe left to right, they may swipe right to left.

Get creative, but don't be creepy, and don't violate people's personal information and space. Create composites of characters based on several different people, or you may get in trouble.

Personas can be a bit narrow, so add more depth to them by creating a story or context around them. One way is to add a narrative and create compelling stories around app

usage.

Set a scene just the way a writer who is telling a story or writing a television show would. Where is the end user? Are they outside? If so, what is the weather and lighting like? Do they have a good network connection or a poor one? Are they inside? If so, where? What is the lighting inside the structure like? What physical characteristics does the user have? Are they a field worker who might have dirty hands from working on physical devices, components and using power tools? How might dirty hands effect usage of the device?

What are the motivations and fears of the user? What tasks, activities and goals are they undertaking with the device when they use our app? How do they know that they have been successful? Do they have a sense of urgency to complete a task? What happens if they miss a deadline, or can't input information or use the app? Do they get punished? Do they lose money?

What other applications and activities might they use in addition to our app? How often will they be interrupted by other apps, communications and physical interruptions from other people or circumstances?

Also take other people into account. Are they making noise and distracting the user? Do they have children who are pulling on their arm to get their attention? Are they people helping by providing support, or are they hindering optimal app usage?

Try to create several credible stories with different urgencies incorporating and emulating different users:

- A technophobe who is slow and measured and fearful of the technology
- A rapid, intuitive explorer – someone who gestures and inputs rapidly (almost like a small child might)
- The impatient interrupter – they will context switch between our app, SMS, email, other apps and back and forth constantly
- The tech savvy user. Someone who is mobile friendly, understands the tech and is comfortable exploring workarounds or figuring out how to make things work. eg. Setting up a wifi hotspot with another device if cellular connectivity is problematic.

Test Scenarios

Cem Kaner's paper An Introduction to Scenario Testing[11] is a fabulous resource for this kind of testing, and I have used it a tremendous amount to help kickstart that kind of work. Cem says, and I have also found that if I only have one test approach that I can use, this is one of the most effective for quickly finding problems that our customers might encounter. He has some guidelines on creating test scenarios:

[11] http://www.testingeducation.org/a/scenario2.pdf

> The test is *based on a story* about how the program is used, including information about the motivations of the people involved.
>
> The story is *motivating*.
>
> The story is *credible*. It not only *could* happen in the real world; stakeholders would believe that something like it probably *will* happen.
>
> The story involves a *complex use* of the program or a *complex environment* or a *complex set of data*.
>
> The test results are *easy to evaluate*.

Cem describes working with real users when possible, and finding out their key motivations and what they value about our software. Ideally, you work with them to create credible scenarios for use, and try out the app based on those stories. You also add in some variation because that's what people do, and no two people will use an app in exactly the same way.

However, if you don't have direct access to users to create scenarios, you can work this out yourself with a bit of research. Look for people in your own social or family network who might use an app like that, and ask them some questions about what they expect. If it is a mass-market app, go to a public place and observe people who are using mobile apps. Strike up a conversation with some of them. If it goes well, they will show you their device, their apps, and tell you what they like or dislike about them.

When you combine personas with credible test scenarios, you have an incredibly powerful tool at your disposal. It takes a bit of pretending, and if you are out of practice, get a book on acting and improvisation. Become the person, and try to solve problems the way they would.

Examples

If I don't have any information about the end users or potential customers, I will create three generic personas:

1. A technophobe
2. A power user (someone who is tech savvy)
3. An app switching user who is impatient

It's pretty simple for me to find three of those people in my life and have them interact with a mobile device.

Technophobes tend to move slower with technology and their interactions with devices, and they often blame themselves when something goes wrong. They depend on good usable designs to be able to perform even basic activities.

A power user gets frustrated when apps perform slowly, if they crash or if they are overly awkward, but they are more permissive of poor design.

An angry user will get frustrated by any delay, any excessive work, or anything that is a bit awkward. If they can't quickly and easily (effortlessly) complete their tasks, they will get angry.

I now add different constraints. One is age. Observe a parent or grandparent on a device. Now observe someone who is younger than you, such as a teenager. Finally, see how a child interacts with the device. They are all different aren't they?

- Grandpa is slow and deliberate
- A teenager is rapid and switches between apps and contexts constantly
- A child will explore by gesturing, and will repeat actions over and over

What kinds of problems might Grandpa find that I will miss? One is timeouts. If he takes a long time to complete an action, the app may reset itself to a known state after a while. How about a teenager? Since they rapidly switch between activities as soon as they get bored with your app, they can cause interruption and interaction problems as they flip between your app, SMS, a social networking app and a game, all several times over the course of a minute. Finally, a child will cause all kinds of state problems due to gesture overload and repetition.

Another constraint is time. What happens when they are in a rush? How does that change how they use the app and the problems you will discover? Try to run through scenarios for each user type, but imagine they are in a big hurry and need to complete the task quickly.

Now add in criticality. What if the activities absolutely have to be correct? Does the user know that they got the

right answer? Is it clear enough for Grandpa to understand? How about a child who can't read? Could they figure out that there was a success or failure?

I also add in emotion as a factor. Try different kinds for each, such as Happy Grandpa, Sad Grandpa, Angry Grandpa, Depressed Grandpa.

Now combine them, and you will be amazed at the problems you will find just by some simple pretending, while you attempt to emulate typical app usage. Things you missed previously testing as yourself will leap out at you when you use scenarios based off of personas. Plus, it's a lot of fun to do.

Test Tips

If I am really pressed for time, I'll just create one in addition to me: the annoying family member. You know the one who gets their PC filled up with viruses and calls you for help? They get a new smartphone, and you spend hours with them helping them get it set up, and then they phone you hours later using their land line because they forgot the password on their screen lock. We all have a friend or family member who struggles with technology, and many of us who are so-called "good with computers" spend family get togethers fixing someone's computer or device.

I either get that person to test the software for me and observe what they struggle with, or I just pretend I am that person, and do things the way they would with the device and app.

Bug Story

I was training a tester who was very experienced with web and PC apps, but was nervous about transitioning to testing mobile apps. One of the first exercises I had her try was the annoying family member test. Since she is the family technophobe, she decided to model her very tech savvy, but impatient son. Whenever she would load a file, instead of waiting for it to load, she would get bored and switch applications. She would send an SMS text message, check email, go to a social networking app, then go back to see if the file had loaded.

Turns out that the app couldn't handle the interruption during file loads, and went into an endless loop of retries. The lesson here is that you don't always have to be that involved with your personas and scenario tests. If you try to change your perspective, you'll probably discover something important, even if your perspective shift isn't perfect.

N: Network Conditions

As I write this, one of the most common sources of errors with apps has to do with network connections. We tend to assume that our apps will have a constant, reliable, consistent and fast web connection. When apps encounter the dynamic nature of wireless networks, they frequently run into performance problems, error conditions, or they just crash.

Mobile apps have different connectivity constraints. First of all, they move around, so they have to switch between different providers and use different technologies. Secondly, wireless communication is not the same as its wired cousin. There is interference, differing signal strenghts depending on how close you are to the source.

Any app that requires the use of wireless technology to connect to the internet for it to work properly will need to have different network types and common mobile conditions tested thoroughly.

Many apps require a reliable, fast connection to a server to work. We assume that we have reliable network connectivity because we almost always do when we are using PCs. On wireless devices, this is not a guarantee. Many apps will crash, hang or behave strangely when they encounter less than ideal network conditions. It's important to test this out, because when people are on the move, network activity, technology and conditions can vary a great deal from location to location, or while we are moving.

Wifi and cellular networks are not as fast as wired network connections for PCs, and are susceptible to a lot of different kinds of conditions:

- Slower or faster technology (different kinds of wifi, different kinds of equipment, and different technologies for cellular data access)
- Interference when on the move from steel in buildings, tunnels, etc. and from other devices or infras-

tructure that can emit electrical or other forms of energy or wireless signals that can interfere with wireless waves
- Transitioning between network sources while moving. (This is very difficult for apps and especially back end systems to handle.
- Wifi can have different signal strengths or faster or slower wifi technology
- Cellular services – different strengths, performance and technologies as well as different performance or reliability from different carriers
- Network transitions – wifi to wifi, wifi to cellular, cellular to cellular, cellular to wifi, wifi to a dead spot and out again, cellular to a dead spot and out again, wifi to dead spot to cellular, cellular to dead spot to wifi

Examples

Earlier in the book described different mobile network types. Here are some of them:

- Wifi
- 2G, 3G, 4G LTE
- EDGE
- NFC

It's important to test all the network types that your device supports. You should also test them out on different carriers, because they will use different kinds of technology, and

will experience a different cadence of busy and light traffic. That traffic pattern will add in different kinds of delays into your app's network communication. There are also different times of day when networks are busier than other times, which can impact performance. One local carrier we tested had problems with outages at noon every day for a few weeks. Utilizing these system outages into our testing was a great opportunity to find new problems, and to test out our error handling capabilities.

Signal Strength

With wifi, it is easy to tell if you have a strong, medium, weak, or no connection, because the wifi symbol has three or four bars. Full strength shows all of them, medium shows two (or three), weak is one. If you have no connection at all, it will be blank. (On smartphones and some tablets, you will need to turn off cellular networks in your device settings to see a blank. If your device has a cellular data plan, it will shift to a cellular network.)

With cellular networks, it can be hard to find out what signal strength is. Research and install free cell coverage map programs that will help you determine that kind of information.

If you aren't using a program to help you determine your connection strength and speed, it will take a bit of trial and error to find locations and times where you have poorer performance. Remember that steel can interfere with wireless waves, so explore the area around where you do your testing, and find the spots that always have poorer

connections.

Also figure out times of day that are busier. It's often Noon and at the end of the day at 5pm when people are taking a break from, or ending work and are using their devices for content snacking or to connect with people. Also identify and utilize dead spots for your testing work. Sometimes it can be difficult to simulate some of these connections technically, but if you know of areas that are weaker because of interference, or times of day that have poorer performance than others, it can be quite simple to get up and walk to a spot that is notorious for poor network performance.

Apps often time out, hang or freeze up, go into unrecoverable error modes or outright crash when they have a weaker network connection than what the programmers expect them to have.

Exploring Changes and Transitions

A few paragraphs ago, I mentioned several different technologies for connecting to the internet. Let's explore three of them from a testing perspective:

- Wifi
- 3G
- NFC

Each of these wireless technologies provide connections to other systems, and they do so using completely different

methods. Inside your device, they each have their own hardware, firmware and software. The device itself manages these connections for you. It tries to provide you with the fastest and cheapest options. If there is wifi available, it will go with that. If it loses wifi and can't find another signal for you, it will switch to a cellular network.

If the preferred method, such as 3G is too weak, it will then move to something less performant such as EDGE. Everything from the technology on the phone to the systems that carriers use and the languages used in the actual communication themselves are different. NFC and bluetooth are completely different beasts altogether.

Now imagine what happens as your device is on the move, or there is interference or network problems. The transitions from one network type to another takes a lot of technology and resources to work properly. You are putting your device through its paces as you walk or move around when you are connected. Now imagine how your app deals with the interruptions and extra device resource use when you transition from one type to another.

This kind of scenario is incredibly common on mobile devices, especially in our "always on, always connected" world. We expect our apps to work, even though the technology to support that is complex, difficult and can be unreliable.

Another thing that can really throw things off are deadspots. This is where you get no cellular connection at all. Some of them, like elevators also interfere with wifi. As we move,

we go through dead spots, and when the device loses all external internet connections, it tries to find something to connect to. Once we leave a dead spot, it goes through a list of preferences and tries to find the optimal thing to connect you to. If you have a good network connection and go into a deadspot, your device has to work hard to try to get you connected again once you move out of it.

I have found countless problems by using an app that required an internet connection while transitioning between different network types and in and out of dead spots. Any large building will cause these kinds of issues, even if you are just transitioning from one wifi beacon to another.

Different Operator Networks

As I have mentioned previously, cellular networks have a vast array of technologies that can be used to reach a designation of "3G' or "4G LTE" and others. Each operator, or carrier will have different types of equipment and technology to support their customers. This can result in different performance, accuracy and reliability from provider to provider. You will probably want to test out your devices on different carriers, depending on what your user community use the most. Furthermore, when we travel and out devices "roam" we use network providers in other countries or areas that have business relationships with our carrier to support traveller use. When we roam, we use a different network than the one at home based on these alliances.

Example Test Combinations

This is a natural area for combining activities and technologies.

Make sure that you aren't testing these areas passively. Work with the app to make sure that it is utilizing a network connection when you make a transition from one signal strength to another, or from one network type to another. If a page has loaded in your app and it isn't doing anything, it probably won't be affected by a transition. If you are submitting information, or search terms, or gesturing and inputting and changing things in the app (watch for the wheel spinning) and you transition from one situation to another, that's where the interesting problems will be found.

Make sure to move from one wifi source to another, and from wifi to cellular networks and back again. Also utilize dead spots.

Combine network tests with your functional and user scenarios. These are great angles to explore to maximize your bug finding abilities.

Test Tips

Find out when the app is using a network connection and set up that moment when you test to transition. Here are some examples:

- Logging in

- Downloading a file
- Sending a message
- Filling in a form and submitting it
- Clicking on a link or image
- Searching for information

You may need to talk to designers and developers to find out when the connections are being made so that you can narrow it down and be the most effective with your testing time.

Utilize common knowledge, so talk to others about dead spots, poor connections, or times when performance is poor for a particular network.

There are network conditioner tools and emulators that can be used, but try this out manually first so you can tune the tools to behave the way you experience conditions in the real world. Many tool defaults assume much better conditions than you will experience in the real world.

Utilize repetition. Try moving back and forth between transitions, signal strengths and in and out of deadspots several times, with different actions. Sometimes an app will fail after several actions, not just one, or a particular action will trigger a failure, where others do not.

Mobile developer John Carpenter points out that communications within an app are run in a different process than the code that manages the user interface. The UI, or what you see on the screen may be waiting for communications

information to come back, but network issues are slowing things down. When these error conditions occur, the other parts of the app that depend on them should have error handling to fail gracefully, or inform the user of options to try to work around the problem.

Bug Story

Recently, I was teaching my Mobile Applications Testing course to a corporate client. One of the attendees got up and walked out of the class while we were working on networking tests on the software they were developing. She wandered off with a tablet, and I could see her disappear through the parking lot and into a park across the street. Twenty minutes later, she burst back into the class room with an inoperative tablet. Right around that time, the systems administrators poked their heads in the room and asked what in the world we were doing.

She had walked away from the office and found a transition point from wifi to a cellular network, and walked back and forth until she found the exact point where it switched. She then begain a file download from the server and stepped across that line.

The app couldn't handle that transition, and failed. It got stuck in a retry mode, and ended up freezing the device. When she restarted the device and launched the app, it would go into that error state no matter what she did. She even uninstalled the app and reinstalled it, but it didn't help.

On the backend, the servers were stuck in an infinite loop of retries and errors, and they were unusable. The dev and test servers were inoperable, and they wanted to know what we had done to take them offline. This was a massive bug, that could be quite common in the real world, but they hadn't thought to test for it until I came along. The admins and developers weren't too happy with me at first, but they came and shook my hand later on and thanked me for helping them save a potentially disastrous problem before they released the app.

Combined Activities with Technology: Devices and Screens

A lot of our mobile device usage is combined with other devices. There is a lot of talk out there right now about "multi-screen" usage. This means that people use multiple computing devices at once, or they combine tasks or goals with several devices. They may use a mobile device while they are using a PC, or when watching television. Here are a couple of examples from my own usage:

- Researching a potential purchase on a mobile device first, then making the purchase on a PC
- Researching television or movie trivia while watching television to enhance my viewing experience
- Texting friends while watching a sporting event on television (usually during Calgary Flames hockey games)

- Reading online forums discussing a certain sporting event I am watching on TV
- Checking email or social media information on a smartphone while using a PC
- Texting or instant messaging on a mobile device while using a PC
- Researching an entertainment purchase (book, movie, game, etc.) on a smartphone, purchasing on a PC, using the entertainment on a tablet
- Posting photos or information on social networking sites on a smartphone or tablet, then editing and improving them on a PC
- Purchasing travel-related services (flights, hotels, etc.) on a PC, and using virtual tickets, receipts and vouchers on a smartphone

Combining activities with different technologies in these examples put a whole new spin on mobile apps testing. You have different goals, motivations, and expectations for usability. In fact, some of the examples above are things I only do because it is easier to type and edit on a PC than a mobile device. However, other activities are more natural. I research and look into things when I'm on the move and have a few minutes to spare. "Hmmmm… I'd better check into available flights for my next consulting gig."

When you are watching TV, you have different posture because you are usually sitting or lounging in a comfortable chair. This changes how you view the screen, and the

angle of your fingers as you gesture. You may also have completely different light to tend with. Furthermore, if you are working with a service or application that has multiple device support, it is important to create scenarios based on people completing parts of tasks on different devices. Is their experience unified, or all chopped up? Can they complete activities by performing different tasks on two or three different devices? Can they pick up on one device where they left off on another?

Combinations are fascinating, and there are many of them to consider when you test mobile apps. There are so many factors to try to keep in mind when testing, and a little research can go a long way in helping you determine useful areas to explore to help you change perspectives.

Tracy's Thoughts

Tracy: I loved the I SLICED UP FUN mnemonic. Memory aids like this are awesome. This especially appealed to me since I have a science background. It's an easy to remember check list to get an overall feeling for the app and an easy way to find bugs related to important mobile factors and how the app handles them. It's a great way to get introduced to a new app, or as a planning tool to help create strategies for a testing project.

I was surprised when I started working with a group of professional testers that I would find bugs that others would routinely miss, just by using a device like this to help me think of test ideas that are important for mobile users.

Concluding Thoughts

Testing within a framework like I SLICED UP FUN can be incredibly powerful. Sometimes the tests seem too obvious - surely the app I am testing won't have those problems - but many testers I work with are surprised at how easy it is to find important problems others miss by testing within a structure. Start out by trying the app or web experience out each of the areas that make sense, then make it more real by combining activities. For example, try location services

under poor wireless conditions, and use a lot of movement, just the way we all use the devices out in the real world.

Chapter 6: Logging and Diagnosing Bugs

The best tester isn't the one who finds the most bugs or who embarrasses the most programmers. The best tester is the one who gets the most bugs fixed. -Cem Kaner

We observe many things when we test, and we distill those observations down to useful information. That information is important to different stakeholders on our projects. We also provide our opinions on the quality of the software we are testing. That information and our opinions help them decide if the software is ready to ship or not. Since we're the ones who methodically work our way through applications (sometimes several times over the course of a release), we become the go-to people for information on how well things actually work.

Managers and sometimes even executives use our information to determine whether software can ship, but other stakeholders find what we produce useful as well. Marketing and sales people will ask the tester how important features work for a sales pitch, and technical communications people will ask us for help as they write manuals. Even slightly embarrassed programmers will pull us into meetings to demonstrate something in software they wrote that we understand how to use better than they do.

Testers are depended on to provide all sorts of useful information, but one of the most important sources of information we can provide are:

Useful and timely bug reports.

If you want to learn more about how to be a great bug reporter, read Cem Kaner's paper: Bug Advocacy: How to Win Friends, and Stomp Bugs[12]

I don't care if you are a grizzled testing veteran, or a nervous newbie. If you haven't read Cem's Bug Advocacy paper (in at least the past month), I urge you to do so. It is full of great advice that will help you create great bug reports. One of my favorite themes Cem highlights in the paper is this: *"Bug reports are your primary work product. This is what people outside of the testing group will most notice and most remember of your work."*

We provide an investigation-driven information service, so there is little tangible that we have to show for our work. As I learned from Cem, the primary method for demonstrating our work is with our bug reports. Sadly, we often do a poor job of it.

When I speak at software development conferences, developers will approach me afterwards. They will start the conversation with something that sounds like a new work lead for me. This is exciting, because that's how I make my living as a consultant, I help software development

[12] http://www.kaner.com/pdfs/bugadvoc.pdf

teams. Then I await their query about implementing a cool concept that I have just spent the past hour talking about.

However, many times they ask this instead: *Can you come in and teach our testing team how to log bugs properly?*

It's a let down. *Really? That's what you need my help with?* runs through my mind.

I'm polite. I like work, but down deep inside I wonder how on earth people could call testing their profession, or do the role temporarily on a project (or even just play a tester on TV for a while), and not spend time working on the *one* visible product we have to demonstrate our competence, ability and talent. Not to mention finding and helping get important bugs fixed is a cornerstone of our job. For many of us, that's our whole job.

I have worked on several open source testing projects, where our fault-tracking tools are open to the world. It's a great opportunity for a tester. You can show your bug reports to people you want to work with. In a job interview, if the choice is between the person who says they are great at logging bugs, and the person who can show you, who do you think wins out? Sadly, my open source project experience, coupled with volunteer work for a professional society for software testing has confirmed what programmers tell me: many of us are horrible at logging bugs.

I was left scratching my head at incomprehensible, unrepeatable, or unclear reports more often than not. I learned to cherish the small number of people who wrote some-

thing useful, and looked forward to their reports. Some of the poor reports were even full of ALL CAPS SHOUTING ABOUT HOW INCOMPETENT THE TECHNICAL TEAM WAS AND WHAT A BAD IMPRESSION THIS PRODUCT LEAVES ON PEOPLE. It was pure joy to read them. (Not.) Do you think that the technical team wanted to deal with these bug reports, and by extension, the people who wrote them? I can assure you we did not.

It doesn't need to be that way. In essence, logging a bug is an easy task to do, you just need to put some care and attention into it. Report what you saw, why there is a problem, and what you thought should have happened. Be specific so someone can repeat it and fix it.

Add media to help - a picture (screenshot) or short video is worth at least a thousand words. My challenge to you, my mobile tester reader is: **be an awesome bug reporter**. Your teammates will love you for it, and you'll prove your value as an in-demand, skilled team member. You can be a great tester, but if you can't communicate *what you observe* in products to others, you won't be effective.

The remainder of this chapter will help you think about how to be better. Follow along, jot down some ideas, and then practice, refine, improve and practice some more.

Now let's focus on being an awesome bug reporter.

Logging Bugs

When you spot a problem, notice something weird, or have a negative reaction or suspicion, you need to investigate further so you can report this information in a useful way. The difference between getting an issue addressed and fixed to help improve the product experience and an issue or bug getting ignored often has to deal with the accuracy of the information in the report. Here are some tips to follow when you notice something that should be dealt with, and how to report it well.

Determine if it is Reproducible

Sometimes we spot a problem, stop and write down the steps it took to reproduce it, and then move on to other things. A few minutes later when we decide to log the issue formally, we can't repeat it. In other cases, we repeat the issue once ourselves, document the steps to reproduce, only to have the bug report sent back to us because the programmer can't repeat it themselves.

We'll talk about how to cope with intermittent bugs later in this chapter, but to reduce the waste in time and energy incurred by logging something others can't repeat, I use Jerry Weinberg's "Rule of 3." When I am filling in a bug report, I make sure I can repeat the bug three times before I start recording it. If I can't repeat it three times in a row, I need to spend more time researching and investigating before logging a report for the programmers.

Some problems are quite easily repeatable, so I move on and start gathering information and logging the bug. Others may occur once out of three attempts. I will then try to repeat it more times, and I may only be able to repeat it five times out of ten, or thirteen times out of twenty attempts. If that's the case, I add the word "Intermittent" to the bug title and include those metrics. That just means that the programmer will have to try to repeat it a few times rather than once to be able to repeat the conditions and start fixing the bug.

Clear Title and Description

I describe the problem in the simplest, clearest terms that I possibly can. This is like the headline of a news story, it needs to be compelling to draw in the reader. Examples include:

- "Poor performance during application launch on iPhone 3GS"
- "Application crashes when submitting purchase form with a weak wifi signal"
- "Incorrect login error message is obscured in French due to special characters"

I spend time on my titles to make sure they are clear and free of typos. It sounds easier than it is, I will often go through several ideas per bug until I get it right. Ii I have any doubts about the clarity of my report, I ask a colleague

for a quick review. If it's clear to someone else, that should reduce work for the team.

I also have a brief description, usually based around a task or fulfilling a goal to put context around the bug. That helps the programmers figure out why they should fix the bug, and what sort of priority it should have. Here is an example:

"While walking between wifi connections, I tried to submit a purchase, but the app crashed. This seems to happen when the wifi signal is weak."

Exact Steps to Reproduce

This is incredibly important. Write down the steps, and practice it yourself. Also explore to see if there are any extra steps that you can remove and still see the problem. If you eliminate one or two steps and it no longer occurs, be sure to also note that in the report. This is useful information for diagnosing the source of the issue.

Be sure to note anything in the environment when you are working with steps to reproduce bugs on mobile devices. Remember that movement activates sensors, different networks have better or worse performance, transitions between network types can cause problems, and low battery can cause the device to behave very differently.

You may notice a problem when you are on the move, get back to your desk and be unable to reproduce it. I always take screenshots with the device (or take a picture with a different device) so that I can make sure I haven't missed

anything important. I also pay careful attention to what I am doing, and how different environments and activities can cause problems.

Once I work to repeat the bug, I will narrow things down to try to eliminate extraneous information, or to uncover truly useful information. For example, I had a low battery when the bug occurred. Does it also occur when the battery is charged? Maybe the wifi connection wasn't at full connection. Can I reproduce it with full wifi? I noticed the bug while walking and moving the device, which gets movement sensors involved. Does it still occur when I am at my desk with the device sitting on it, without moving? In other cases, I might have been using a cellular network. Can I repeat it when I'm using a more powerful connection like wifi?

Understanding these issues helps me create a much more accurate bug report. There are more dimensions to be aware of with mobile experiences, so pay careful attention to what is going on with the device, within the environment, and with you.

What Should Have Happened

Testers have great ideas on how apps can provide value to customers and users. We often use these motivations as we generate test ideas. We notice that there is a problem, but it may not be obvious to everyone else. Furthermore, the programmer may only have a vague idea of a better solu-

tion, so our ideas will help them as they solve the problem, or as they confer with designers and other developers.

You can also find information in specs, requirements documents, or in developer guidelines that are supplied with each mobile development framework program. This can help provide credibility to your bug reports. Check out the Human/Computer Interface guidelines that development frameworks provide for more ideas.

Device and Environment Details

At the minimum, always include the following:

- Device manufacturer and model number (eg. HTC One X)
- Operating system version and sub-version (Android 4.0.3)

Notice how specific that is? I don't say: "An HTC Android phone" or "HTC One X on Android 4", because they are too vague and there are too many other variations to look into. You can find this information in the settings app on your device. Also be sure to include anything about the app, including the version of the software you are testing eg. *Project Xena 1.01 Build 24*.

Here are some device details to consider including when relevant:

- The date and time when you noticed the problem
- Wireless network type
- Low battery (can cause device to slow down)
- Movement and sensor interaction
- Using location services, such as GPS or assisted GPS
- Other activity on the device: other apps running, interruptions by OS, communication, etc.
- Apps your app depends on, such as web browsers
- Device heat (can cause device to slow down)

Here are some environmental issues to include when relevant:

- The type of wireless connection (wifi, cellular, others)
- Network connection strength, or if a transition between two networks occurred
  - (Note, with cellular networks you may need to get a program to help get that information)
- Lighting conditions
- Temperature and weather
- Device movement, user movement, and travel speed

Supplementary Information

The old saying that a picture is worth a thousand words holds true for bug reports. Despite my best efforts, I may still have a poorly worded bug report, but an image or

video capture demonstrating the problem may save the day. What I was unable to describe I may be able to demonstrate perfectly using media.

One of the most difficult pieces of supplementary information for beginners to gather are stack traces. Stack traces are complex error messages that are often suppressed from the user interface. You see a friendly message: "Oops, sorry, an error occurred" but in the background, more specific information is recorded and stored. To get this information requires development tools, so you will need to work with your friendly neighborhood programmer to figure out how to get this information. It's worth the extra effort though, this information is incredibly helpful for programmers.

> Mobile developer John Carpenter says: "This. A million times this. A stack trace is worth more than a case of beer to developers. A stack trace can often show the error to the developer without the need for trying to reproduce the bug. This makes for faster bug fixes and better code and products."

Spend time with technical people to learn how to gather this information that they find so useful. It's worth it!

Reduce Duplicate Bugs: Search the Bug Database!

When you are working on larger teams, or on software that has had several releases, it's a good idea to search the bug database when you are ready to log a bug. Make sure that what you are logging is new, and you aren't logging a duplicate of a bug that has already been recorded. Getting a lot of dupes in a system is frustrating for other team members, and it is easy to lose track of important information. If you find a bug that someone else logged, just add your insights and information to it, rather than logging the same bug again. Duplicate bugs cost time and effort to track down when they have been logged, and team members grow to dislike testers who cost them valuable time.

Tools to Help

Mobile devices are great for capturing media, such as images and video. We utilize them constantly with other applications, or simply to record our holiday snapshots, a pet in a silly pose, or what we are eating for dinner. Why not incorporate them into testing? They are an enormous part of mobile content creation for social media and other uses, and they can be easily used to help bug reports. Videos are fantastic to show movement or other problems that are related to the use of sensors or environment conditions.

Cameras

Mobile devices have image and video capturing built-in, so utilize it on the device you notice the problem on, or if it is easier with another device, record a video or take a picture from a separate device. You can even create a video of someone else reproducing the bug, especially if there is movement or other environmental factors that are required to reproduce it. Screen captures of errors, particularly error messages that are difficult to reproduce, can be a godsend to a programmer who is trying to figure out what went wrong.

Development Tools

To get more information, such as the exact version of the software you are testing, and stack traces or other error messages, you will need to use either a development tool, or a configuration or device management tool. Both tool types enable you to pull information from the device that you won't usually see through the user interface. To use them, you plug your device into a PC that has the tool, and once the program syncs with your device, navigate until you find the program information. A stack trace might start off looking like this:

```
FATAL EXCEPTION: main
java.lang.RuntimeException:
    at android.app.ActivityThread.performLaunch
```

The key with these tools is to scroll through the log file in the device management tool, find the time when the bug occurred, and look for "Error" or other key words that indicate a problem around the time you noticed something odd. Copy all the error text, and some of the text before and after the error in the log file for good measure, and include it in the bug report.

Mobile tester Johan Hoberg points out that sometimes you need the developers help to get a special build which enables additional debug logs or enables the use of specific tools. If you point out a problem, and there isn't enough information in the stack trace, a developer may be able to create a special build that has more logging and debugging information for you to try out. Ask if this is possible, because they may not think of it at the time.

Inside a Programmer's Mind

When I get a bug report and need to decide whether I can fix the problem or not, the first thing I look at is the title and summary. Can I figure out what the problem is? Is it clearly stated, or is it some vague or nonsensical issue that doesn't make any sense? If I understand what the issue is, next I look for

steps on how to reproduce it. If they are vague and I can't figure them out, then I can't fix the problem. I have to repro the issue on my development machine, preferably with debugging tools so I can find the source of the bug. If I can't reproduce it, I can't fix it.

In many instances, a great set of repeatable steps makes up for a vague or incorrect title and description. Next, I look for supplementary information to help narrow things down so I can start solving the problem and creating a fix. I look for stack traces (detailed error messages that have important code, 3rd party library and time-related info), screen shots and video. Sometimes, I can't repeat the error, but a stacktrace gives me the right information on where to find the problem in the code on my own. Other times, I can't repeat the problem on my machine until I watch a video recording of the bug on your machine. Then I notice something different that I had missed out on.

If I have a great title, great steps to reproduce (without a lot of extra steps that aren't needed) and great supplementary information, that makes my life easier and I can deal with the problem. For extra points, add in suggestions on how to make it better. I'm not omniscient and I need your ideas too.

Sometimes, I can repeat a problem by following the steps, but I have no idea why this is an issue. The person who logged the bug didn't tell me *why* this should be fixed. If you don't care whether I can fix the bug or not, and make me feel like I'm wasting

> my time, then don't provide clear steps, be as vague as possible, and don't provide any supplementary information. I will read the report, feel frustrated and assign it back to you as "can't repeat" and ask for more information. However, if you also want to make me angry, then lecture me in the bug report. Or scold me for creating the bug in the first place, or provide me with "helpful" information about how I can do my job better. Remember that it is difficult for any of us to receive criticism about our work. (Even professional testers!) A little kindness and empathy can go a long way.
>
> Know that just like you, I am busy and stressed out, and reading bug reports does not rank highly on my list of favorite things to do. If I get a lot of really negative, unhelpful bug reports, it can be depressing and kill my motivation. Don't be negative, but collaborate with me and be helpful. Fixing bugs can be a thankless, difficult job, and we can either work together and get them sorted out easily and quickly, or we can be adversarial and get nowhere. In the end you, me and the customers all benefit or suffer, and it can stem from something as simple as a bug report.

Here is an example bug report from Tracy:

Sample Bug Report

Reported By Tracy Lewis

Date Reported 6/2/2012

Program Name frombluetored.net

Configuration iPad 3 16 MB

Can Reproduce Yes

Severity High

Problem Summary Magnify causes strange flashing behavior when using a pinch or spread gesture

Problem Description and how to Reproduce

Steps:

1: Access the book by entering frombluetored.net in the browser

2. Tap the "All systems report Ready! Click here to proceed…" button

3. Press Play and try to magnify the book by spreading your fingers over the screen

Results:

The iPad goes into a succession of flashing pages from the book (see video for demonstration)

Expected Result: You should be able magnify the book for easier reading without errors or strange behaviour.

Video attached: gesture_bug.mov

Diagnosing Bugs - Dealing with Intermittent Problems

Intermittent or so called "unrepeatable" bugs are problems that are difficult to reproduce. Intermittence is tricky with mobile devices because there are a lot of factors that change depending on location, timing, and where and how they are being used that can create different conditions. On many software projects, if an intermittent bug is difficult to reproduce, they are called "unrepeatable" and forgotten about until they cause problems for customers. On mobile projects, intermittence can occur much more frequently because of the extra complexity inherent in the devices:

- They are used on the move in different locations with different lighting and weather
- Network technology varies from place to place, and changes as you move
- The devices have built-in features such as sensors and optimization tools
- Location services technology is varied and used with many apps
- Platform fragmentation of device types, operating systems, helper programs, cellular carriers and countries is large
- The "always on", "real-time" information and communication create constantly changing conditions

Remember earlier in the book when I asked if you could figure out the why a bug occurs 5 feet away from you, but doesn't occur where you are sitting now? Let's generate some possible ideas for why that might occur:

- You were standing and holding the device in a different orientation
- Device movement (even slight) was engaging sensors
- The device was connected to a different network (different wifi, or a cellular connection)
- You were standing in an area that had weak network signal, in an transition between two network sources, or in a dead spot
- Wireless interference from steel or other devices
- The lighting was different, causing light sensors to engage
- Your device was in a low battery state, and returned to your desk and plugged in the device to charge it
- The device was hot, and went into heat dissipation mode
- You interacted with and inputed into the device differently - fingers at a different angle, or you were faster or slower
- Something on the device was syncing or updating information or a program and it isn't anymore
- There were more apps or other services and features running at the same time

These are all plausible theories, and some of them relate to a different location, but others relate to timing. Another factor to consider are the emotions of the user. Emotions such as impatience, anger, nervousness, fear can have a big impact on how we interact with a dynamic device when we are happy or feeling more neutral. We may shake the device or move it more, we may use different gesturing motions - hard and fast when impatient or angry, less accurate when nervous or fearful.

To try to repeat an intermittent bug, like the example we have discussed: "it happened over there, but doesn't happen here!" you will need to try to repeat what you were doing when you saw the problem, taking those and other aspects into account. You would get up, move to the exact spot you were in, and repeat exactly what you were doing. Hopefully, it will be an obvious and easy issue dealing with location. If it is due to multiple factors, or factors that aren't easy to spot and re-create, you will need to be more creative.

Avoid Intermittent Bugs by Paying Closer Attention

Intermittent bugs are frustrating because they can take a lot of time to diagnose and narrow down. Reduce the chances of your own intermittent bugs by taking note of various conditions whenever you see a bug. Either commit them to memory, or record them in a note. Here are some dimensions to be aware of when you spot a bug to help

repeat it later:

- What is the exact location where you found the bug?
- Were you moving around or doing something that could engage input sensors?
- Were you moving? Stopping and starting frequently?
- Were you holding the device differently than you usually do?
- What network were you connected to?
- What was the signal strength?
- What was the battery strength like?
- Did you feel heat from the device?
- How did you discover the bug? What are the steps to reproduce?
- How were your emotions? Were you in a hurry?
- What is the frequency of occurrence? How often does it occur? For example, 3 out of 10 attempts to repeat, or 8 out of 10.

Become a Software Investigator!

Back in February 2006, I attended a peer workshop on software testing hosted by Cem Kaner. We each had an angle or idea to present, and one of the things Cem talked about was the parallels between software testing and professional investigators. In fact, Cem has a fabulous definition of software testing in The Ongoing Revolution in Software Testing[13]:

[13] http://www.kaner.com/pdfs/TheOngoingRevolution.pdf

> "Testing is a technical investigation of a product, done to expose quality-related information."

This resonated with me: a technical investigation is interesting and requires skill. There's none of this passive "go verify that we met the requirements" limitation on our work.

Cem also mentioned investigation in popular culture and how we can apply lessons we learn in our leisure time to our testing work. There is a long history of detective novels, television shows, movies and computer games that we can draw from. I'm willing to bet that many of you enjoy TV shows that are based around investigation.

Now think about your favorite fictional investigator, and imagine how they might approach an intermittent bug. Here are some classic investigator tools:

- Observation (use the 5 senses)
- Interview people who might know something useful
- Look at photos, video, related documents
- Get access to records or logs
- Scenario modelling (what could have happened?)
- Hunches, intuition (knowledge + experience), feelings

Great testers learn how to develop each of these tools, and related skills. Next time you have a testing problem, think about how your favorite investigator would tackle it, and broaden your horizons. As you develop your skills, learn how to observe better, ask better questions, and to find information that might be otherwise hidden from you. And always trust your instincts.

Intermittence Tips

Here are some things I do to try to get a repeatable case for intermittent bugs.

Generate Theories to Explain Intermittence

The first place I start when I try to find a repeatable case for an intermittent bug is to come up with a plausible explanation. If I don't have one, I talk to other people including designers and developers, and especially the person who saw it last. What did they see, and do they have any theories about what might cause it? I may start out with several ideas, and try the all out one by one, carefully observing whether my theory is plausible or not.

James Bach recommends that you don't hold to strongly to your pet theory, and that you aren't too influenced by other people's theories. It's important to keep an open mind, and use evidence to support or refute any theory, even if it's yours, or comes from someone important on the team.

My first idea (or ideas) on how to repeat an intermittent bug, and why it might occur are well-researched, thorough, plausible, and almost always wrong. However, as I apply rigour to my investigation and test out my theories, I notice new information and that helps me adjust and adapt to hone in on the real problem.

Take Careful Notes!

Use your notebook or favorite electronic tool and note everything you can regarding the issue. Write down ideas, observations, theories and hunches. When you interview people, note their names, roles, and the date and time. When you test, write down your observations. If you see patterns in screenshots, video, log files, or other similar problems that others have logged, write them down. After you have exhausted your initial ideas, combing through your notes can be a great source of ideas and inspiration, and you may notice something you missed in the past.

Also, share your observations, ideas and theories with others. Collaborating and combining different perspectives frequently helps track down a nasty intermittent problem.

Thorough Observation

Take note of the issues I noted earlier in the chapter, including how the device is used, where it was used, what the environmental conditions were like, network conditions, device states etc. Try to look beyond what is obvious, and

expand your frame of view. Pay attention to what is going on in the device, environment conditions, location, and movement.

Interviews

Talk to the people who have seen the problem. If necessary, get them to re-enact the same situation. With one hard-to-reproduce mobile bug, I asked the product manager and a sales person to re-enact their sales demo to me, including where they were standing, and where the participants sat. They thought I was a bit nuts at first, but they even went through the part of their presentation leading up to the problem and pretended I was a potential customer. Something amazing happened - they used the device completely differently during the re-enactment than they did when they described the problem. I realized that sensors and extra gestures were causing the device to freeze up.

Also ask others for their theories on what might have gone wrong, and if they have seen anything similar in the past. Carefully record their thoughts and ideas, even if they sound outlandish. Don't judge and don't lead them towards your pet theories - get information and keep your opinions to yourself. Any idea is worth considering, and you want their ideas, not the ones you have already tried and aren't getting anywhere with.

Take Pictures, Video and Look at the Log Files

It is amazing how a screen capture or video recording of the last time a bug occurred can provide new information that you previously missed. To look under the covers, find out what the log files recorded the exact time the bug occurred, and show it to a developer. This is often a great way to figure out issues that were timing related and not related to anything else you were doing with the device. Note: Often, there isn't any obvious information in the log at first, such as ERROR BUG HERE! so noting the time when it occurred is vital.

Model Scenarios

Famous detectives often solve cases by coming up with a credible scenario, and then re-enacting them. Earlier we talked about creating user scenarios. In this case, we want to think about worst-case events. What would happen if multiple failures occurred at once? Or what about if multiple events happen at the same time? Bounce these ideas off of designers and developers, and ask them about their ideas for worst case scenarios. If they describe a scenario, but qualify it with "But that can never happen so don't worry about it…" then you know you are on the right track. As John Carpenter points out: "In developer speak this often means: 'I never considered that scenario and have no idea what the code will do in that case.'" As testers, we can perform an enormous service to the people who

haven't considered alternate possibilities and outcomes by testing them out. Don't get put off just because they display resistance. Prove it out with evidence through testing.

Most weird problems happen under "that can never happen" conditions. Telling me that a failure like this could occur under circumstances is like throwing raw meat to a junk yard dog. I'll thank them for the idea, and run with it. If I can prove through evidence that in fact it *can* happen, then we can build a better product. The "that can never happen" scenarios are an absolute gold mine of test ideas.

Change Your Mind and Repeat

Throw out theories when they don't yield results, and modify them. Consulting tester James Bach has two popular investigative mnemonics: "OFAAT" and "MFAAT". OFAAT stands for "one factor at a time" and MFAAT is "multiple factors at a time." If I can't quite repeat a problem, I will reach for MFAAT or OFAAT, depending on what I am doing. I'll reduce factors, or control one factor to help narrow things down or get more information. In other cases, under simple conditions, I will add more factors and see if the rate of intermittence changes.

When you have exhausted your options, review and see what you might have missed. Try alternate theories, add them to your original ideas, and slowly introduce more variation.

Be a Scientific Thinker

Early in my career, a tester named Tim Van Tongeren mentioned that the testing approach we were using and talking about publicly had its roots in the scientific method. Tim talked about the difference between quantitative research and qualitative research in scientific research. Quantitative research involves repeating experiments over and over to see if you get the same results. Sophisticated programs are run to calculate whether an experiment is repeatable, and by other people, and a weight is applied to it. I ran into quantitative experiments when I was an undergrad. I could make a small amount of money or get bonus marks in a course if I took part in an experiment. These were run by grad students, and they didn't like it very much if your results differed from what they were expecting.

Qualitative research on the other hand has to do with experimental design. This is where the interesting discovery occurs. The scientist or the professor is the one who leads this effort. Once they have something interesting, it then moves to other people to try to repeat it and generate the metrics.

Personally, qualitative research interests me much more than trying to repeat a bunch of things other people have done, especially on software projects. I like new technology and learning about how it works within our lives. I decided to research qualitative research, and I re-discovered the story of Ernest Rutherford, and how his discovery led to the creation of modern nuclear physics. (Note, this is a

very simplified gloss over of this story. Science-y people, I apologize!)

It all hinged around a model of the atom. At the time, the popular model of how an atom was composed was the so-called "plum-pudding model". To demonstrate and explore the model, scientists could fire alpha particles at a gold foil, and observe how the particles passed through the foil. Rutherford performed this experiment, but noticed that it didn't always work out as planned. There were particles that seemed to get lost or deflect away.

It would be hugely tempting to ignore this intermittence because it didn't fit the model that was popular at the time. I don't know if you have worked with scientists, but they don't really like it if you don't agree with their ideas. If you're going to contradict them, you had better have some good, hard data from a reputable source.

It's also easy to repeat something many many times to make anomalies statistically insignificant. I have witnessed this with grad students who were frustrated when my attempts to help them out with their experiments went horribly wrong. "Well, if we re-run it 1000 times in ideal conditions, we can bury Jonathan's weird result and ignore it" is what I imagined them thinking before they kicked me out of the lab. (Joke's on them, now I get paid to create experiments that go horribly awry.)

Rutherford knew something wasn't right, so he decided to adjust the experiment to capitalize on the intermittence instead of ignoring it. Instead of one foil, he set up several

around the area where the electrons were fired, repeated the experiment and made careful observations. Those observations led to the discovery of the model of the atom that we use today.

On software teams, intermittent bugs tend to be ignored. They don't fit the popular team model that the software is awesome and without fault. If you repeat the exact steps for the intermittent bug every time instead of adjusting to capitalize on the intermittence, and think why it might be intermittent, you probably won't find the source of the problem.

Rutherford's method underpins scientific discovery, and using that sort of thinking when testing (which is a lot like an experiment)helps us discover important information about the software we are analyzing. The important discoveries in science are due to finding something weird, adjusting our experiment, and observing the weird from a better perspective.

In science, we don't predict and figure out things very well the way it is often portrayed in popular culture, like the lone inventor myth. We blunder about, see something that doesn't fit, adjust and learn.

Too often in the industry, we only focus on a style of testing akin to quantitative research - we create and repeat tests over and over, instead of the discovery rich qualitative research, which is about experimental design. I don't want to repeat things over and over unless I absolutely have to. Instead, I want to adjust and adapt and discover and

learn, and share that information with the team. The more variation in my testing, the more I will discover things that people using our software might uncover. Life and the environments we use mobile device in are varied, so it's important to incorporate that into our testing.

Dealing with Resistance

When we get bad (or merely critical) news, it isn't what we want to read in an already stressful day. Life would be grand if everyone complimented us on how smart, handsome, perfect and wonderful we are. But, we are human after-all, and we need constructive criticism to improve ourselves, our work, and on software teams, our product. As testers, our primary work product is often bug reports, which are all physical evidence to the contrary of the perfection of our product and our designers and developers. As you might imagine, various stakeholders might not be that happy when we log bugs, particularly if they are at inopportune times.

We can't help it though, we have honestly and clearly report what we see, and get that information to the people who need it. There are some things we can do to help though.

Here are some common patterns I have observed, so you can understand *why* others on your team might resist your bug reports, or try to write them off and close them without

fixing anything. As I wrote in my Test Automation Politics 101[14] article:

> "When you are the bearer of bad (but realistic) news, don't be surprised if someone is disappointed."

Challenging Delusions of Grandeur

Sometimes, people on our projects have some unrealistic ideas about how effective their tools and processes are. They think that if you use a tool or follow a process, there should never be any bugs, ever. So far in my career, I haven't found a bug-free project, and I don't think it's a bad thing. Software development is complex, software operates in complex environments, and we depend on a lot of other people's programs, code and services. Something is bound to go wrong somewhere.

On mobile projects, we often run into delusions because people oversimplify the project:

> It's a smaller project, (after all, the screen is smaller) so we should have far fewer problems!

[14] http://www.kohl.ca/2009/content-pointer-automation-politics-101/

> New tools and processes should be faster AND bug free! After all, that's what the tool salesman and process coach told us!

Over the past fifteen years, here are some ideas people have told me would produce bug-free projects:

- The C++ STL (standard template library)
- Object Oriented Programming
- Software inspections
- The Java programming language
- The Model View Controller design pattern
- Agile software development processes
- The Test-Driven Development programming style
- Automated unit testing
- Automated testing in general

Each of these were also used to tell me that testers would no longer be needed on software projects. So far, I am fifteen years in and I haven't seen anything bug free yet. Each of these approaches may *alter* my approach to testing, but it doesn't get eliminated. There are still problems that need to be discovered, feedback required, and quality information to help make shipping decisions to be made.

Encountering Zealotry

Some people get an idea in their heads that things should work the way they expect, and they don't want to deal with

anything to the contrary. That innocent little bug report may stir up passionate feelings on a par with politics or religion. My wife likes to joke that I need to come to parties with a warning sign: "If you talk to me about politics, religion, or software development, you WILL get offended."

People get really set into ideals with software development, and they can get quite religious about their preferences. This can involve processes (Agile vs. so-called waterfall), tools (Java vs. Objective-C), devices (Android vs. Apple iOS), and the granddaddy of many of them: text editors (vi vs. emacs.) People can become enraged if you challenge their belief in the superiority of any of those issues. These also extend to issues on our project, and your bug report might upset someone.

There are also other pressures on our projects. Budgets are an interesting one. In some cases, managers or other stakeholders are very concerned about costs. They might see bug reports as a threat to the budget, or they may see them as a threat to a tool that they stuck their necks out to buy. They might feel that your bug report might put their decision to buy that tool in a poor light. "This tool cost a lot, so it has to work!"

There also might be pressure from a vendor or a potential partner. The ideal of the projects or tools working together seamlessly and watching the money pour in for everyone could be threatened by testing that proves otherwise. There is nothing like money, or the potential for money to get people excited. If there is a threat to that ideal (even just a

perceived threat) people might freak out.

Blaming the Messenger

There might be a thousand problems on a project that stakeholders are currently coping with. Then, in comes your bug report. They might react like this:

> Argh! Not another problem!

They aren't upset with you personally, but they don't like that you are now adding to the burden of the project. In reality, you're doing your job, but they may not react well to you because now there are more problems.

In many cases, you just might not have enough credibility and trust with the team yet. When you are new to the team, or you're a hired gun tester, you will face resistance at first because they haven't learned to trust you yet. But, you will be awesome and they will learn to love you and your bug reports, even if they don't know it yet, or don't show it very well.

Dealing With Resistance

As I mentioned earlier, be careful when you log bugs to not accuse people, and to log them accurately with lots of detail to help people fix them. However, even when you do your best, you might still face resistance.

- Don't take it personally.
- Use evidence, not emotions to show how it needs to be fixed.
- Don't make people feel stupid.
- Don't blame people for the problems, we all *maek misteaks*.
- Don't use bug reports as platforms to further your own pet process, practice or tool changes.
- Don't play politics to get your way.
- Demonstrate alternatives for improving the product.
- Be patient, resistance fades in the face of evidence and logic.
- Don't get overly attached to a bug, or a set of bugs. There may be very good reasons why they aren't addressed in a particular release, so if they aren't fixed right away, let it go.
- *Don't be an overly negative jerk.*

I can't stress the last point enough. Remember to try and find some positive things to encourage your team members with. Having to fix a bunch of bugs is a tough job and isn't much of an ego boost. If someone on the team relishes in other's mistakes or lectures them, it gets old really fast.

The Coke Can and the Soldering Iron

I'm going to close out this chapter with a story about intermittence and taking hardware and sensors into account.

See if you can identify my advice in this chapter with what I describe here. Did I miss anything important in my approach?

I was just out for lunch with a friend of mine. He works with mobile devices that are custom designed and developed for vehicles. He was a few minutes late because he had been informed of an intermittent problem they were having with devices that were attached to certain vehicles at client sites. Some devices stop working temporarily, and they can't figure out what was going wrong. The problem is intermittent, and only occurs in the field. They have never seen this behavior in the test lab, and when devices are analyzed in the lab, they work perfectly. He asked me for ideas so that they could track down the intermittent bug. I asked what was different about these vehicles, and he said they move more quickly than others, and they vibrate a lot more. I then asked about heat.

Are they used in the hot sun? It turns out that they are. My advice was to try to recreate the problem in the lab, and I'd start by taking a device, shaking it and heating it up. I'd try to do this over a period of hours, even if I had to borrow a paint shaker. If that didn't help us cause the problem, I would look at networking, moving the device between network sources, and find out any other information that might help track it down. That's how I approach recreating problems that come from the field - emulate field conditions in the lab and get a repeatable case that the programmers can fix.

As we were chatting, it reminded me of a story that I thought I'd share with you. Several years ago I worked for a wireless company that specialized in remote data management. We had custom-built mobile devices that were small computers hooked up to sensors, with a modem and antenna, and a battery with solar panels for charging. These devices were set up to measure water or oil and gas pipelines, or anything else that was in an out of the way place with stuff that needed to be measured. They would send the data using different wireless technology. Since it was difficult to access these areas, (sometimes they were set up in conflict zones), you didn't want to visit the device unless you had to. So they were set up to last for years, with operating system and software updates being sent over the air (OTA), much like how our smartphones and tablets get updated.

If a device failed in the field before its projected end of life, it could cost a client a lot of money to fix. We had heard reports of an intermittent bug that would cause these devices to go offline, requiring a trip to reset them and get them working again. We had tried to replicate it in the lab, but we weren't able to nail it down. In fact, only one tester saw the problem at all. The rest of us never witnessed it, but she would see it a couple of times a week, and would call us all over. No matter what we did to try to replicate it, we couldn't. It was intermittent.

I was just starting out as a testing consultant at the time, so I was charged with tracking down a repeatable case. I tried

different theories, and nothing worked, and then I decided to watch how the other tester worked, just in case there was something I was missing. I noticed that they did a lot of OTA testing, and they had a lot more hardware on their desks than the rest of us. In fact, they had the embedded device sitting on top of a couple of other hardware devices. With their permission, I spent a morning testing using their test bench and PC. As I typed, I noticed that the device would rock. When I stopped typing, the device would sit still. I then checked the cables that were hooked up to the device, and they were seated loosely. One of them, for network, was practically falling out. That gave me an idea, so I went back to my test bench, loosened the cables, and started running tests on the device. To simulate the rocking motion, I pushed my knee up and down on my table while I was going through different tests.

I was effectively combining activities. A device could vibrate in the real world due to wind, or other physical effects in the environment, and devices don't sit there doing one thing at a time. I remembered that this tester loved to do OTAs, so I fired up an OTA job, and then started rocking the table with my knee. Almost instantly, the device froze up. On a smartphone or tablet, we would say that it was bricked. It couldn't do anything without a hard reset. I practiced this several times, and I could repeat it every time. When I showed the lead developer how I could repeat the bug, he looked at me like I was crazy when he saw my knee going up and down, but once he figured out the logic, he laughed and found a fix right away. In the field, if a

device had poorly seated connections, and was vibrating, or was subject to intense heat (direct sunlight in with high temperatures), and the cables weren't seated fully, while an OTA was being performed, the device would freeze up. Once we knew the cause, it was simpler to repeat in the lab, we just used to a tool to make the network connection to be weak and lossy while doing an OTA update.

That started us on a path of doing more physical things with the units. Later on, we were having trouble repeating a problem that was caused by a device temperature alarm in the field. We were having problems figuring out how to process alarms that were sent from the devices over the wireless network, and we didn't have any test data for it. Customer support had asked clients for alarm codes, but the general feeling was that unless someone shared some codes with us, we wouldn't be able to fix it. I thought otherwise, and said:

> "This sounds like something we need to emulate in the lab."
>
> "But we're talking huge temperature extremes here! How do we do that?"

I went to the kitchen, and grabbed a can of Coke from the pop machine, and put it in the freezer. Then I went to a test bench, and got the biggest soldering iron I could find. I

found a temperature sensor, and got some help from the lead hardware dev on how to hook it up. Then I spent time setting up a tool to capture all the wireless traffic from my device. Once the can of Coke was near freezing temperature, I brought it to the test bench, and I fired up the soldering iron. I held the probe to the soldering iron, and alarms started flashing past on the traffic monitoring tool on my PC. I let the sensor cool, then held it to the frozen can of Coke. Again, alarm codes flashed by.

The lead developer knew how I worked, and once he had seen me create the codes manually, he created some custom test sensors for me by soldering potentiometers onto a wire lead. Now we could create extreme temperature ranges, and we could change the temperature instantly without having to wait for an object to be cold or hot enough. We were able to repeat the problems with the turn of a switch, and we were able to create all kinds of test data for future use.

That is how I used a soldering iron and a can of Coke to help repeat an intermittent bug. When you are working with physical devices, things like location, temperature, light, network availability and interference are vital to being able to track down intermittent problems. Mobile devices blur virtual and physical worlds, so don't be afraid to add something physical

Tracy's Thoughts

Tracy: With bug reports, the devil is in the details. It may seem obvious but a clear title, exact steps to reproduce and a screenshot or video are really important. Give as much concise information including device and environment details so the bug can easily be replicated. Developers and designers can't read your mind. Also offer suggestions on how a change to the software could eliminate the bug for easier use. They don't have all the answers, so feedback on *how* to make the software better is also helpful for them. Using screenshots will show information that we might miss, and videos are important for any bugs that require movement either due to inputs, sensor interaction or you moving around to different places with the device.

And be nice!

Concluding Thoughts

Bug reports can make or break your testing career. If you consistently create great reports that take mobile dimensions into account (device state, environmental conditions, wireless conditions, context of use and the people using it) you'll be much more valuable to your team. Many mobile bugs can be difficult to reproduce because people don't

pay attention to factors beyond the functional steps or activities they were doing at the time. Also use screen captures and video to help record bugs - what might be difficult to describe can be easily demonstrated by using the technology mobile devices provide us by default. In fact, as you use these technologies along with the app or web experience you are testing, you might find integration bugs or other problems unrelated to what you were testing at the time. Bonus!

Chapter 7: Mobile Test Strategy

"Strategy is the great work of the organization. In situations of life and death, it is the tao of survival or extinction. Its study cannot be neglected." ~Sun Tzu

Strategy: Figuring out What and How to Test

When I start out testing a new mobile app, the first place I start is to create a strategy. Mobile projects are unique in that they tend to have extreme time pressure, and they are highly visible products. In other words, we don't have much time, and if the application is poor, a lot of people will see it and complain about it. (They usually do it publicly, using application store ratings and social media.) Mobile apps also have a lot of visibility internally within organizations. One client told me that if their e-commerce system was down, there were fewer complaints than if they had a mobile-related problems: a mobile website outage, or a poor mobile app release.

So what is a strategy?

A strategy is your guide for reaching an objective.

What is an objective?

An objective is a clear goal, or target you can reach.

It is easy to squander energy when testing, ie. spending too much time on low-value activities on any project without a good sense of priorities. This is particularly easy to do on fast-moving mobile projects, so take a little bit of time to think about how you are going to make sure your time is spent effectively. Thinking about a strategy will help you decide how to make the best use your time, resources, tools and techniques. Your strategy can be brief and informal (in some cases you might not even write it down), or it might be detailed and formal (written and shared with stakeholders).

This chapter has two parts. In the first part, we examine how to decide what platforms to test on a project. This is a common issue on mobile projects because there are so many devices to choose from.

The second part explores creating a test strategy for your project.

Part 1: Platform Testing Strategy

One of the most common questions I hear when people are contemplating a mobile testing project is: *What mobile platforms do we test on?*

This is one of the most difficult challenges we face on mobile testing projects. Emulators are fine for *very* basic

look and feel and limited functional testing, but you're going to need to do testing on the real thing, whether your focus is on automated or manual testing. The real devices have sensor and touch interaction, as well as network and other physical aspects that need to have real-world testing.

Notice that I don't just name devices, I use the term *platform*. That's because there is a lot more than just purchasing a device and testing it. There are several factors to consider. This is how I define a testing platform:

A mobile device platform is a combination of:

- Hardware (a smartphone or a tablet model from a manufacturer)
- Operating system version (most have sub-versions, eg. Apple's iOS *n* will have several versions over time)
- Networking:
  - Wifi
  - Cellular networks providing wireless broadband
    - Different telecommunications companies that use different technology, operate in different countries, and often have special software they install
- Web browser (if you are testing a mobile web app or site)

At the time of writing, we have a huge market of hardware including smartphones, tablets, e-reader devices, gaming

systems, and mobile features in televisions and vehicles have navigation, telephony and internet access in their on-board systems. We are now seeing the emergence of smart watches, and other wearable and sensor-based computing platforms are on their way. When testing, you will need to consider the following:

- Both smartphones and tablets
  - (Plus any other relevant devices)
- Different hardware and operating systems or versions, per framework
- Devices with different screen sizes
- Different wireless types (wifi, cellular, different signal strengths)
- Different cellular network carriers

Each of these considerations can yield different results when testing. An app may depend on communication features that are part of a smartphone, but is missing that feature on tablets. Different hardware may have different performance, or may support different sensors and services differently. An app may have been designed for a larger screen size, and doesn't display well on smaller screens. For example, I frequently test with a smartphone and a tablet at the same time, and it's amazing how the app can behave or appear differently on one or the other. Different cellular providers can have different performance levels, and some may be more reliable than others.

Device usage is also incredibly important to factor in, and will help you narrow your focus to activities that are more suited to the people who will be using your app. You need to understand the people who are using the devices, and where they use them.

1. Understand the target customer base:

- Who are they?
- How will this app create value for them?

1. How do they use mobile devices, where do they use them, (in buildings, outside) and in what weather conditions?
2. What mobile devices do they use? Do they use cellular networks for network access, or do they stick to wifi?
3. I look at the following questions to help guide what I should purchase for a test lab:

- What platforms are our customers using?
- What are the most popular smartphones and tablets on the market?
- What are the most popular mobile web browsers?
- What platforms are known to be problematic? (aka. problem child devices)
- When is the newest thing coming out? Do we need it for testing?

The answers to these questions can help you narrow down your testing, or expand it out beyond your test lab.

For example, if your target customers only use Android devices from one manufacturer, and they always use them when they are outside or at a client site using a cellular network, that narrows down your testing focus. You need to purchase the devices from that manufacturer, install the popular operating system versions, set up cellular data plans with the carriers they will use, and go outside and test under various weather, temperature and wireless conditions.

If your application is for the mass market, then you have a more difficult problem to solve. You will need to purchase more devices, and test under more even conditions.

Platform Testing Strategies

You are going to need to test on as many different physical device combinations that you can. It will be impossible for most of us to test on *every* device/platform combination, so we have to choose what to pick for testing. We have constraints, such as limited budgets, limited people, and limited time, so we have to be careful and consider our options.

I have four different strategies for dealing with this problem:

1. Singular Approach:

We only test on one device type. This is either because that is all our team plans to support, or if we are under extreme time pressure, I can only choose one. This might be the most popular device in a device family, with one operating system, using one cellular carrier, one cellular network type in addition to wifi, in English only. If I am forced to only choose one due to time or equipment constraints, I'll pick the problem child device - the device that is a bit slower, and that programmers had more trouble supporting. That way I can find more problems quickly related to performance or other issues that they may not have had time to test themselves.

2. **Proportional Approach**:

If I don't know exactly what to focus on because we aren't targeting one specific, narrow combination of hardware, operating system, carrier, language etc. then I need to do research. I find out what the most popular platforms are in the market, and how those fit within our target market and what our team is willing to support and develop for.

I have to do a lot of research with a proportional approach. There is no one web site or service that can provide me with the exact information I need to determine what the most popular devices are in our target market. I have to do some research and figuring.

Here is one way I approach this: I start by looking at mobile web traffic on our web site. Mobile traffic doesn't tell me the exact hardware device that people are using, or the sub-version of the OS. If I'm lucky, I can get information such

as iOS, Android, Windows Phone and Blackberry access, as well as a web browser for each. Now imagine that we have 50% Android mobile traffic on our corporate web site, 45% Apple iOS mobile traffic, and the remaining 5% are other handset types.

That can help us narrow things down. At its simplest, we would then spend 50% of our testing on Android, 45% on iOS, and the remaining 5% on other platforms. To simplify further, we can just decide to look at Android and iOS.

We'll look at iOS in an example later on, so let's just look at Android.

Android has a fabulous resource that shows the current distribution of their versions of their operating system: Android Developers Platform Versions[15]. I just looked while I was writing this paragraph, and Android Gingerbread versions 2.3.3 - 2.3.7 have about 57% of the distribution, while Ice Cream Sandwich versions 4.0.3 - 4.0.4 have about 21%. (By the time you read this, I'm sure it will have changed.)

This information helps us narrow things down. Clearly we need to look at both Gingerbread and Ice Cream Sandwich operating systems, and possibly spend more of our time testing those than other versions.

An alternative approach is to research reports on the web, and find out what is most popular in the market, and then work from there.

[15] http://developer.android.com/about/dashboards/index.html

We can research market conditions and mobile usage further to find out more information using services such as Our Mobile Planet[16] from Google, Akamai IO[17], Net MarketShare[18] and others.

3. Shotgun Approach:

This is for a mass market app or web app. We need to support all sorts of devices, and we have no self or customer-imposed restrictions on devices. It's wide open, so we need to try to test on as many different platforms as possible. This depends on a lot of research to do well. It has the highest risk, because there are many, many platform combinations out there, and we will undoubtedly miss a particular combination that doesn't work well with our app.

I spend a lot of time searching the web for reports on current device, operating system and cellular network stats. I try to find as many pie charts and graphs as possible, and I deduce what might be the most useful for us to test to provide the most value from these reports. If 3 out of 4 reports are recent, and agree with each other, I will use that information together to help narrow my focus.

I also look for problem devices, both by talking to programmers, and searching the web. "Phone Model X sucks" is a good place to start. You'll find programmers ranting

[16] http://www.thinkwithgoogle.com/mobileplanet/en/

[17] http://www.akamai.com/html/io/index1.html

[18] http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

on development forums about a device, and they are often cheaper, less powerful, less common devices that are quite inexpensive to add to our testing array.

I also look for information from people who are creating device labs to get their thoughts and views on what devices to purchase for testing.

Designer Brad Frost has a really solid post Test on Real Mobile Devices without Breaking the Bank[19] and so does Dave Olsen How to Build a Device Lab[20].

Search for other opinions from designers and developers, they will have valuable insight to help you figure out where to go.

4. **Outsourced Approach**:

There are various services you can use to supplement your own test devices with basic testing on devices that other people own and have set up. Formally you can do this work with remote device access services which allow you to install software and control a device remotely over the web to do basic functional tests. You can also use a crowdsourcing service where they manage people with different device types in different locations and parts of the world to do testing on their phones for you.

You can also harness your own personal networks, either

[19] http://bradfrostweb.com/blog/mobile/test-on-real-mobile-devices-without-breaking-the-bank/

[20] http://www.dmolsen.com/mobile-in-higher-ed/2012/06/26/how-to-build-a-device-lab-part-1/

in real life, such as friends and acquaintances you see regularly, or asking people you work with to use their devices to test, especially when they travel. You can use social media to orchestrate simple tests virtually, or you can meet your friends in a restaurant and have an "Apps with Appetizers" session. If you get a group of people with different platform combinations together, pay for the meal and drinks, and have them go through some common scenarios, you can extend the reach of your test lab from time-to-time.

Here is another way to look at platform testing strategies for coverage:

- Singular strategies:
  - The most popular device or family
  - The "problem child" device
  - The single platform your target customers use
- Mixed Strategies:
  - Proportional: * What our customers use * Low cost multiple device (this is all we can afford)
  - Shotgun approach (as many as possible)
- Outsourced strategies:
  - Remote device access services
  - Crowd sourcing services
  - Utilize your personal network
    * Friends, family
    * Social networking
    * Bug hunt parties: "apps with appetizers"

Steps to Determine What Physical Devices to Use

This is the path I follow every time I work on a testing project, whether it is simple, or complex:

1. Determine the approach
2. Research, research, research
3. Use deduction and best guesses to determine device popularity
4. Map out the platform support problem with a classification tree
5. Determine an optimization strategy
6. Work with stakeholders to see if our strategy is feasible
7. Rework as needed

Let's work through an example together in the next section.

A Platform Strategy Example

Our app is targeted for Apple iPhones, using the iOS 5 operating system. We can use a singular strategy.

Now it's time to research. First of all, I need to understand what _hardware) versions of the Apple iPhone are available. I do a little searching on the web and determine that at the time of writing this book, the following iPhone devices are the most popular and fully support iOS:

- iPhone 3GS
- iPhone 4
- iPhone 4S

We have three hardware devices to consider.

Next, I need to understand what versions of the iOS operating system are part of iOS 5. I fire up my trusty web browser, and type "iOS versions" in my favorite search engine. Wikipedia has a good page on iOS Version History[21] and at the time of writing this book, the following iOS 5 versions have been released:

- iOS 5
- iOS 5.0.1
- iOS 5.1
- iOS 5.1.1

That leaves us with four versions of the OS to consider.

Next, I need to find out if our app requires a network connection, so I talk to the technical people on the team and find out that it does. It will require a connection to work, so we need to take wifi and cellular networks into account.

To keep it simple, we'll just focus on wifi and 3G, since those are the most popular options for network usage on smartphones locally.

[21] http://en.wikipedia.org/wiki/IOS_version_history

That leaves us with two networks to consider.

Next, I draw out these combinations with a classification tree. I do this because our brains aren't always that great at figuring out combinations of things, and there is nothing like a visual representation to help us count it out.

iOS 5 classification tree

After my tree is mapped out, I end up with the following:

To support iOS 5 on iPhones, with both wifi and 3G, we have **24 ** combinations to test. Whoa! Wait a minute! That sounds like a lot! But, that is the reality.

Let's look at the numbers some more: If you add two carriers for 3G (wireless broadband), it goes up to **36** combinations.

If you add iPad (also three hardware versions) to testing, all of these numbers double. We end up with **48** combinations.

Also factor in network strengths, such as 2 or 3 different measures of network connection strength, and it goes up considerably.

This example is probably the simplest, but even with just the iPhone on one OS version, you have **24** combinations just with different hardware, OS versions and two network types.

We can't test all of those! Not to mention, we probably don't have to. So next we come up with an optimization strategy.

Apple pushes updates out to devices, and they are hard to ignore, so it is *reasonably* safe to assume that most people will upgrade to the latest version. However, these get rolled out over a period of time, and some people hold off for a few weeks. We could just choose the latest OS version: iOS 5.1.1 and only test on that. However, if the release is only a few days or weeks old, we'll probably also want to test on the previous version: iOS 5.1.

Let's see how the "only use the latest OS" optimization strategy works. If we take two iOS versions into account, we are down to **12** combinations. That is more reasonable. We could reduce it further and decide to only support and test on the latest and greatest OS, which would then reduce it to **6** combinations.

We can search for "iOS 5.1 upgrade stats" to help us decide if this is a good strategy or not.

At the very least, we have to go through a reasonable amount of coverage 6 different times on different platforms. That in itself is a lot of work.

Now take a platform with more combinations of hardware and operating systems into account, such as Android. The problem is more difficult. There are several OS versions that are out there, at least 3 are popular. You have several manufacturers with several different models of hardware.

There are several web browsers (I know of about 5 off the top of my head.) Now add in the factors we looked at above, and it can get pretty crazy. Mapping out and communicating your findings is vital to helping figure out what to test and not to test.

Once I have an optimization strategy, I talk it over with stakeholders. Many of the other people on the team will have no idea how complicated platform testing is, so I use diagrams and back up my work with facts and information.

If they don't agree with my approach, I can work with them to come up with something feasible. However, if they work with me to cut more platforms from our strategy to save time and money, I advise them of the extra risks we introduce when we optimize.

Your optimization strategy is not foolproof!

Whenever we optimize and narrow our focus, we run the risk of missing something important. Always have a mitigation plan, and encourage your team to plan for surprise platforms that are problematic, but weren't tested thoroughly enough, or at all.

Part 2: Creating a Test Strategy

This is a process I go through whenever I create a strategy:

- Determine your testing objective(s)
  - Risk mitigation vs. reward exploitation
- Perform Analysis
- Examine Resources:
  - Time, people, budget, tools
- Determine your strategy
- What are the limitations and constraints?

We'll explore each of these steps in more detail.

Step 1: Determine the Objective

Test objectives and strategies are rarely handed to us, we have to figure them out for ourselves!

Unfortunately for us, that objective is usually provided in the form of: "here, go test this please" with little specifics. "Test this" is a very poor objective, we can't really tell if we have done a good job or not. So we need to do a bit of research and decide for ourselves on how to make that actionable and specific, and then share that with our stakeholders to see if they agree or not.

On testing projects, I usually encounter two types of objectives:

1. find important problems quickly
2. evaluate software based on its suitability for a particular purpose

The first item is usually encountered when we are testing software that is being developed. The second is usually encountered when we are trying to decide whether we should purchase or use software that is available to us. We may be trying to make a purchase decision, or evaluating the usefulness of a free, open source tool. This will give you a place to start. One heuristic to help identify the actual objective is to look at the type of project. Are we developing new software? It's probably some form of #1: find important problems quickly. Are we evaluating software someone else created because we want to use it? Then we are probably looking at some form of #2: evaluate software for suitability.

What is the difference between the two? In the first situation, we are looking for problems. Our focus is a negative critique. We do whatever we can to find problems that our users might encounter and report them to the developers so that they can get fixed before we release the software. This is more of an *offensive mode*. In the second case, we are looking more at a positive critique. While we watch for problems, we look more at whether the software can handle typical usage. This is more of a *defensive mode*. We want to protect our organizational interests, so we observe and evaluate the software to make sure it fits our purpose. Does it do what the brochure and sales demos claim? Can we work our way through typical usage scenarios cleanly and easily? Does it meet our requirements? The software may have severe problems in certain areas, but if we are unlikely to use those features very much, and there are

suitable workarounds, we may not mind.

There are exceptions to this heuristic though, so you need to be sure you talk to your stakeholders and get their buy-in. Here are two examples of exceptions:

1. Testing newly developed software for a sales demo. We tested to make sure that sales demo scenarios worked cleanly. This was a shift to a defensive mode when we had been working in an offensive mode for the release.
2. Security testing software for purchase to ensure it met local privacy laws and corporate policies. This was a full-on offensive mode during our evaluation.

The key point with your objective is to ensure you have something measurable that is aligned with the important stakeholders on your project. If we just used "test this" as a strategy in either of those exception cases, how would we have known if we spent our time correctly? In fact, I took over testing in each of those scenarios because the testing teams were getting it wrong. In the first exception, they were in full on risk-based testing mode, and were far too slow to respond to sales requests, jeopardizing the company's revenue stream, and the future of the company. In the second exception, testers were too casual in their approach, and jeopardized a project and contract negotiation.

Your objectives will be unique, and as you get more practice, you'll get better and better at helping a team determine

them. An objective should be specific enough so that you can tell whether you have achieved it or not. As I said before, "test this app" isn't a very good objective. First of all, there isn't a qualitative measure here, you can test it poorly, or you can test it well. But if you tested it, you aren't really going to evaluate the quality of your work, and think about if you are missing something important.

If you are told: "go test this app", try to explore what a better objective might be by asking questions. Start with the two above, and work from there. The people who ask you to test the app may expect you to figure out what to do, or they may not have thought about aspects that are important for this test project. Here are some further questions you can ask about what is important to them:

- Do you want me to find important bugs quickly?
- Would you like me to test out common user scenarios?
- Are you concerned that the application works on different kinds of handsets?
- Do you have customers in various parts of the world who are on different telecommunications carriers, speak different languages, etc? Or is this a local app?
- How do you feel about this release? Are you confident? Do you have any fears or misgivings?
- What is the result of a poor release? What about the inevitable "1 stars" on app stores and online complaining you will get? Is it a high profile, high

risk if this release is poor, or is it more important to get features out in the market?

You may end up with different objectives, and that's fine. They also might change over the life of the project. Software development is complex, and our testing projects are often complex as well. Just make sure they aren't too numerous, and if they are contradictory, try to find out why. You may end up with alignment and remove the contradictory parts, or you may end up having to do a subproject within a project to cope. That's exactly what I did for that sales evaluation objective in my exception example above.

Our roles are important. Our jobs are like insurance or safety features - people don't care much about them until something goes wrong. Determining a good objective for your mobile testing work is the first place to help make your work more meaningful, because at some point, someone will care.

Step 2: Analyze the Device Support Problem

Here are some helpful areas to help with analysis.

1. Research the users in your target market. What are their motivations and fears? Understanding why they would use the software you are testing will help

you determine important areas to test. If it is an app that helps make lives easier, then ask: Can it solve their primary problems? If it is for entertainment, ask: Will it be enjoyable to use this software?

2. Research known mobile weaknesses for each platform you are going to test - leverage the knowledge and work of others to help you find things quickly.
3. Make sure you have enough information to learn how to use each platform well so you can make the most of your testing time.
4. Research project documentation: any requirements or specifications, any public claims or statements, and ask about sales motivations and targets. Anything that is written about or spoken about the project will help you figure out important areas to test.
5. Find out what mobile features and affordances are going to be used, and what technologies will be employed. Then research any known problems or weaknesses with those technologies.
6. Research the current mobile market. What are the characteristics? Are there any upcoming changes that we may have to deal with while we are testing?

Once I have a compelling set of answers to those questions, I look at my objective, and start thinking about ways to focus testing, and how I can prioritize. What are the most important areas in the above points that we should look at when we are testing? What is critical? What is optional?

The key is to be able to be an active participant in these decisions. If you come prepared with your testing perspective, you can help the team make better decisions, rather than being a passive participant who gets told what to do, and ends up with an impossibly large task on your hands.

Step 3: Discover Available Resources

Resource discovery is incredibly important. We can start out with lofty goals and ideas on how to approach testing, but this is where many decisions are made for us. What do we have available to help us test? This can take some research to determine. Here are some areas to look into:

1. *People.* Who is available to help?
2. *Deadlines.* When do we need to have finished our planned testing work?
3. *Platforms.* What devices do we have available? Do we need to purchase anything?
4. *Tools.* What testing tools do we require? Do we have PCs, productivity tools, automation or testing assistance tools etc. available?
5. *Space.* Do we have test labs, office space, desks, etc. available to utilize?
6. *Budget.* Do we have money to purchase new devices as they come out on the market? How about to pay for telecommunications and data plans, software tools and other licenses on an ongoing basis?

7. *Schedules.* Are the above resources available when we need them, or are they being used by other groups at the same time?

If I have one person, a deadline of next week, and no tools other than my personal device, that strategy will look very different from one where I have a dedicated team with a test lab full of tools and a decent budget and several months of work alongside the development team.

Step 4: Determine a Strategy

Whenever you test, your job is to use your limited time and resources to generate ideas, execute tests, observe and document what happens to the application you are testing, and provide that information to the people who need it. While that sounds simple, software can be used in many ways, in many environments, for different purposes by different people with different handsets. There is a potentially limitless universe of options to test out. It is up to you to find an optimal path, and that's just what you'll do.

You have two choices:

1. **Choice One**: Ignore thinking about strategy and therefore, implicitly create a strategy.

That means you don't really think about it and just dive in and test. If that's your choice, you'll likely miss important

information and problems. You may do what you have always done before on testing projects, or ask someone else to tell you what to test. You can be passive and hope you stumble on the important problems, (and win that lottery) or you can be active and strategic about how you spend your time.

1. **Choice Two**: Think about and explicitly create a strategy to maximize your time and effort and find out important information at the right time.

An explicit strategy is always a better way to go. In business school, we were taught that up to 80% of new business ventures fail. Another metric they drilled into us was that 100% of business ventures that don't plan will fail. Even though plans change, or were completely wrong, that act of planning and figuring out how to use your resources is an important part of a project. It helps you focus your efforts towards high-value activities. The first step in a solid plan is to create a strategy.

Here are some questions to ask to help you form a strategy:

- What is my objective for this testing project?
- What can I use to try to reach that objective?
- Is the objective clear enough so that we can tell whether we have reached it or not?
- How do I prioritize so I can focus my work, especially if I run out of time?

I used a military quote to start this chapter because the many of the best examples of strategy that I have found are related to military campaigns. Talking about the military isn't popular these days, but there is a long history of success and failures that we can draw from. A testing campaign isn't that far off. We have a lot of pressure, limitations and constraints, and an objective to reach.

I also study game theory, because it is a formal process that looks at decision making, and has mathematical models to help validate it. Game theory has identified two types of strategies: *Singular* strategy and a *Mixed* strategy. Singular has a core focus that you don't deviate from, while a mixed strategy incorporates utilizing different approaches to reach an objective. In military terms, an example of a mixed strategy is *divide and conquer*, which means you wage a battle on different fronts to help weaken your enemy.

Example Strategies

Mixed:

- Testing Tours
- Combined activities
- Divide and conquer
- Attrition - starve resources until failure

Singular:

- Focus on one user

- User objective based testing
- Focus on one thing:
  - security
  - privacy
  - accessibility

Test strategies are funny things. If you don't create a strategy you are doomed for failure unless you are very, very lucky. Teams that create strategies tend to do much better than teams that blunder about and hep to stumble on the right solution. If you create a strategy, and you get it wrong, it doesn't really matter as long as you are flexible and adjust when faced with the reality of a project. I'm not sure why, but teams are rewarded for putting in the time and analysis work when they put in the effort to create a strategy, even if they have to change mid-stream, or even several times over the life of a project. It's less important to have a perfect strategy than it is to have some sort of strategy, so don't worry about it too much at first. You'll learn to get better with practice.

A word about Tactics Tactics deal with operational activities that help us reach objectives of our strategy.

Examples:

- Share information with others
- Critique each other's testing work
- Use tools to help extend our reach

- Change focus when new risks or interesting information appears
- Add in some variation in test execution

Step 5: Critique your Strategy

What can go wrong?

Mobile projects tend to have the following challenges:

- Mobility is a fast paced and rapidly changing technology and ecosystem
- Massive device and technology fragmentation, so there are almost always too many options for us to test
- Much of the technology is new, and we can face unreliable tools and products that can challenge our initial assumptions, and cause more work than anticipated
- User behavior and expectations are different than with PCs. There is far less patience, and they have public platforms to complain about our apps.

Concluding Thoughts on Strategy

I want you to think about military strategy. While it may not be familiar to you, you will most likely recognize some famous military figures. They are still household names, though some of them have been dead for over 2000 years. Do you recognize any of these people?

Alexander the Great. Ghengis Khan. Sun Tzu. Atilla the Hun. Joshua and David from the Bible. Julius Caesar. Napoleon. Ulysses Grant. General Lee.

I'm sure you have heard of at least some of these names, if not all of them. They are famous for their victories in battle, and for their brilliance in military strategy. Many of them created strategies that are still taught in military schools today. They are still taught because they are still effective, when applied to a modern context. They also often have memorable names, like "divide and conquer", "shock and awe", "flying wedge", "human wave" and "pincer ambush." A brief study of military strategy has helped me enormously as I plan how to attack a mobile testing project.

A more modern military concept is called the OODA loop, which was developed by the brilliant US military strategist John Boyd. OODA stands for:

- Observe
- Orient
- Decide
- Act

An OODA loop is used to create large scale strategies in a war or conflict, but also at the individual level. It's a good heuristic to remember. You're under pressure, so don't just flail away blindly, but stop and observe. Next, figure out what is going on, and where you fit into the current

scenario. What is going on, and what do I know from my own experience and knowledge that fits this situation?

Now that I have an idea, (or ideas) on how to move forward, I pick what I think is the best approach, and I act on it. Can you see a parallel between this concept and my story about mobile strategy above? This is exactly what I did, but I had the luxury of time on my side. I had a whole week to figure out how to act.

Here's another story. As I write this book, I am mentoring my friend Tracy Lewis to be a mobile tester. She works on mobile testing theory, and works on her assignments to apply the theory. (In fact, she is testing all the content of this book.) Theory and assignments are fine, but she needed something real-world. A mutual friend had written an e-reader application that needed to be tested on an iPad, so I suggested that they employ Tracy.

Trouble was, the release was in a week, and there was little time for testing. In fact, Tracy only had an afternoon to test. She decided to get the most of her time, she would use a mixed strategy. That involved using a combination of mobile tours and heuristics like I SLICED UP FUN! as a guide, and work through as many of those approaches as she could. Within two hours, she had uncovered over a dozen bugs. Some of them were known issues, but she found important problems other people had missed. She blew people away with her skill and creativity, all by testing on one platform and letting a strategy guide her actions.

Now look at the OODA loop again. Tracy did exactly that. In the face of extreme time pressure, she calmly figured out what might be the best strategy for success, then followed it. She could have easily jumped in without planning and relied on trial and error, and spent the afternoon without finding any important bugs. Because she had a systematic approach, and wanted to use many approaches rather than one, she didn't let herself get stuck in one low value area for a long period of time. This experience led to glowing real-world experience, and paying work.

One last thought on OODA: one of my friends trained to be an EMT (Emergency Medical Technician.) They are the people that show up in the ambulance when there is a medical emergency. He told me that the best tip he had when he was being trained was to look at his watch as soon as he arrived on scene to a medical call, or an accident. You arrive, get out of the vehicle, look around at the scene, then stop and look at your watch. It sounds like a pointless waste of time, but it was useful because it helps you focus in a time of high pressure and stress. When we see an emergency situation, or instincts kick in, and we get a rush of adrenalin.

While most of us listen to that instinct and are tempted to run away from an emergency situation, they are the people who are paid to go in, and fight that instinct and panic. Looking at your watch helps you focus, and put those other feelings to the side. You see a horrific scene, and your adrenalin kicks in and your brain starts firing a

million ideas a second. To counter that so you can actually be effective, you stop. Look at your watch. Figure out what time it is, and calm your brain down. Then you assess the situation and decide what course of treatment to follow to best suit the situation. This is OODA in action.

Now, if you still think you don't have time to come up with a strategy, read this again. If people who work in emergency services strategize in situations where every second counts, and precisely because every second counts, then you have time to strategize as well, and you need to spend that time wisely.

But We Don't Have Time For a Strategy!!

Ah yes, the time excuse. I get it. We have extreme time pressures on mobile projects. Since the devices are smaller, intuitively, stakeholders think that everything else should be smaller too, including our development and testing time. But, that isn't true, and with mobile projects, testing can take up a large amount of our development time. On some development platforms, I have heard of numbers approaching 80-90%, due to all the platform combinations, and coping with unreliable new technology-related issues.

That said, we may have an hour to test, a day, a week, or even a month. Whatever time we have allocated, it won't be enough, so we have to make some tough decisions on how to focus or people, skills and resources to find the most important information that we can. We had better spend at least a little bit of time figuring out how to best use that

time, because an hour, a day, a week or even a month are quickly used up, and what if you focus on low value items and miss important problems?

Recently, I was asked to lead mobile testing efforts for a smartphone application. There were previous releases, but there were some serious problems with the app that seemed to occur during sales presentations, or during beta test use by potential clients. This was threatening the success of the project, and the problems couldn't be fixed, the whole project might be scrapped. No one wanted to scrap the project, it was cutting edge and there was an enormous potential for publicity for the company.

There were even people from the news media who wanted to do stories on the company, and a mobile app would showcase in that very nicely. My job was to co-ordinate testing for the upcoming release, and test the new code that had been developed to fix the issues that were occurring in the field. This application was suddenly very high profile.

One issue in particular was that the device would completely freeze up, often requiring a restart. Trouble was, this couldn't be repeated in the lab anymore, but every release it showed up in the field. There were some other reliability issues that couldn't be repeated in the lab, and the devs had done their best to address them by hardening up their code.

To start off, I looked at what had been tested previously. I found reams of test cases that were written for PC and web apps that were copied and pasted and adjusted. They were all functional test cases and had very little mobile-

specific information in them. There was nothing about hardware integration such as sensor usage, low battery, using accessories, interactions with other apps on a device, or testing on different network types using different wifi beacons and cellular networks, or scenarios such as testing on the move, or in locations with different lighting, dead spots, or different times of the day.

I started researching known weak spots with mobile devices, and I combed developer forums for problems they encountered with that platform while developing apps. I researched the devices themselves and looked at different options that you could change on each device to personalize it. I then created different categories of testing, or coverage models to bring us beyond functional testing basics. (I also scrapped all those useless test cases that were providing no value, but that's another story.)

I used my research about the product, the market and typical usage, and the mobile space in general combined with different models of coverage to create a *divide and conquer* strategy. In military terms, this strategy relies on a combination of approaches and techniques to reach an objective. You maximize your potential for reaching a goal by using various approaches, which can overwhelm an enemy. In our case, the enemy is ignorance, and hidden information, so we maximized our chances of finding important information by using different perspectives and approaches within the time we had available.

We had a very good project outcome. We found repeatable

cases for the field issues, and we found a ton of other mobile-specific problems that were elegantly fixed by the programmers and designers. The app not only performed flawlessly in sales demos, but beta testers raved about how stable and useful the app was. This app became an important foundation in the sales strategy of the organization, and got all kinds of free publicity and street cred in the industry. That wasn't because I am a special tester, it was all in the strategy.

I was under extreme pressure to just test the way they always had, and to dive in immediately. I stuck to my guns and spent a week researching and coming up with a strategy. I then made the strategy visible to team members, and co-ordinated our testing activities according to it. We could have easily spent the three weeks of testing we had available to us re-running the same old test cases from previous releases, but we would have missed important problems, and the product's future would have been in jeopardy. We would have found the same kind of information, and not figured out how to repeat most of the tricky field bugs in the lab.

Even if I have an hour to test, I will spend time on a strategy first, because the less time I have to test, the more focused and effective my time needs to be.

Tracy's Thoughts

Tracy: Determine your objectives by asking your team members questions. It's important to prioritize testing in case you run out of time, so get their feedback on what you think is important. Try to cover as many bases in the little time you have. Mnemonics are also good analysis tools because they help you be more thorough to understand what should be tested. I use I SLICED UP FUN to help gather information when I create test strategy.

First off, which of the letters in the mnemonic apply to this mobile app or website? I find out by talking to other team members. For example, if the app doesn't use any networking, I can scratch "N" off the list and not worry about testing in that area. If the remaining letters work with this app, then I make sure to prioritize which are the most important, and make sure I spend time testing in each one, with more effort in the more important areas.

I've seen others spend far too much time testing in one area, for example, only functional tests, and miss other really important areas for mobile testing altogether. I like to make sure my strategies support varied approaches. Even if I spend less time in one area than others, I find important problems quickly because I test a lot of different things. The mnemonics are good places to start.

Concluding Thoughts

A strategy is one of the most important factors on successful test projects. Mobile projects require a tight focus and a way to actually address the potentially limitless amount of testing we might face. They also need to have flexibility as a component because things can change rapidly. New operating systems, devices or wireless technology can disrupt releases, so if your strategy takes change into account, it won't be so disappointing when it happens.

Until I discovered that the test strategy is the real key to managing a great testing project, I was discouraged when I created test plans that no one seemed to find useful. James Bach's work was incredibly encouraging - a plan is a snapshot of what we are doing right now, but a strategy really guides our activities. It is so vitally important to strategize, I urge you to try it on every project you're on.

Chapter 8: Guidance and Planning

"I always felt if we were going in to do an album, there should already be a lot of structure already made up so we could get on with that and see what else happened." - Jimmy Page

There's a reason I quoted a musician in the intro of this chapter. When I was younger, I played in a few different bands. There are two that I remember well. One was made up of a mix of skills from highly structured and disciplined experiences. Three of the four of us were classically trained musicians. Three of us had been involved in sports and up to the college level. There was a high degree of musicianship in this group as well, but another band I was in also had two professional musicians with many years of experience. On paper, you would have expected the second band to outperform the first. However, it did not, because we lacked structure. We had fun, we enjoyed our time, but we were not productive at all. The first band was incredibly productive over a short period of time. In fact, some of my friends still ask for recordings that we made over fifteen years ago.

With creative work, structure might seem like a drag on productivity, but constraints allow us to focus our energy

in high-value areas. That's what Jimmy Page was talking about in the quote I used to open this chapter; use the structure to build on. Without it, you might get lucky. Or you will more likely waste a lot of time (and money) rediscovering the basics of structure and have little time left over to be creative and innovative. The constraint that structure puts on us can be looked on as a burden, or an amazing boost to productivity if used correctly.

Here are two common patterns in software testing to *avoid at all costs* on mobile projects:

1. Testing with no structure or planning at all, and hope you luck out and find important problems before your customers do.
2. Spending too much time creating test documentation up-front (like test plans and test cases) leaving little time to test beyond what's in the docs.

Both of these approaches are irresponsible. The first pattern doesn't have enough structure, while the second one has too much of a restrictive structure. There's a balance between too little structure, too much structure, or even using the wrong structure altogether! Make sure the structure fits the technology, the people and their context. Traditional QA practices were developed in the 70s and 80s, and sensor-based computing such as smartphones, tablets, and other wearable computing devices like watches and bands require a more modern approach that fits them better.

When we test without structure, we often leave unpleasant tasks to the end. That's human nature. We focus on things that are comfortable and easier. We also tend to repeat what we did most recently because it is fresh in our minds. (This is sometimes referred to as the "recency effect".) Unfortunately, that rarely translates into thorough testing in areas that are important to test. If we aren't systematic and thorough every time we test, we're taking a huge risk.

On fast-paced mobile apps projects, you may only have a few hours or a couple of days to test. If that's the case, and you don't have documentation requirements, you can structure your thinking using mnemonics like I SLICED UP FUN! and mobile testing tours. You can simply use these in real-time as you test to help you be thorough in your approach. These are both tools that can be used as mental checklists. I have seen a few hours of testing structured with relevant I SLICED UP FUN test ideas and a handful of tours outperform several days of testing that only focused on functional aspects and spent a lot of time on testing documentation.

On the other hand, you may have much more time for planning and thinking about how to approach testing on a project. If that's the case, use those tools to help guide your planning activities so you can make sure you are focusing your efforts in the right areas.

There needs to be a balance between having no structure and guidance, and something that is too heavyweight and restrictive. Mobile projects are in a fast-moving sector of

technology, so you have to be careful about what you write down. If you record a lot of heavy documentation, it will probably need to get rewritten next week. On the other hand, you also need to focus testing energy in the most productive way you can. If you just do what you feel like doing, you'll probably miss important problems. There's a balance to be struck: using mobile-friendly guidance and structure to help manage our efforts and report results to stakeholders.

Adding Structure

There are some simple ways to structure your testing approach. One is to plan out areas of focus. You can write this down formally, or you can use it as a thinking guideline. Here is a simple example:

- Installation testing
- Workflows and feature testing
- Scenarios (general use, get out in the real world)
- Relevant mobile testing tours
- Device interactions with:
  - The Web
  - Wireless networks and transitions
  - Accessories (power/syncing cables, stylus, etc.)
  - Other people, apps or devices
- Documentation review
  - Within the app
  - Marketing materials

- Uninstallation testing

This is *one* approach to testing an app. There are several models of coverage there that can be explored to various extents, and it covers the lifecycle of an application on a device with installation an uninstallation testing.

Another way to add structure is to model the application to get different perspectives on areas to test beyond the specifications or requirements. Identify common workflows, list all the features, understand typical usage, and different user-controlled options that the app employs. This can help change the focus in either of the above examples of models of coverage. It helps you focus testing on areas that exist in the app, instead of just testing areas by rote. For example, it wouldn't make much sense to perform network testing on an app that doesn't use it, or to test uninstallation with an app that is pre-loaded on a device.

Understanding the users, and how and where they will interact with and use the app can also change the focus entirely. Imagine the differences between testing an app designed for grandparents and one designed for teenagers. The way they interact with apps, their expectations, and what they find valuable can be very different.

Guidance documentation can be incredibly useful to help focus testing in areas of high value. Some mobile-friendly examples include coverage outlines, use cases, checklists, and quality criteria.

We'll explore some of these devices in more detail in this chapter.

Don't Overdo It!

A note of caution: testing has a lot of folklore about how projects are supposed to be run. Traditional folklore includes legends, traditions and fairy tales, and software development is also full of these sorts of effects. Do not implement practices for structure and guidance on a mobile project just because that is what you have always done, or what others have done in the past. Mobile projects are fast-moving and change a great deal, so be careful to be thoughtful about the type of structure you impose. It needs to reflect the technology and be flexible to allow change when changes are forced on you and your team by the market and technology advances. Mobile devices and the systems that support them are complex, and usage is incredibly varied, so encourage variation in testing to reflect that.

One of the chief complaints I hear about traditional testers on mobile projects is that they don't adjust to the technology, and spend a lot of time on activities that don't provide a lot of value. That includes spending a lot of time up-front on test documentation, and not spending enough time on test execution. Once they start testing, they focus narrowly on black-box functional tests, or requirements verification, and don't get out in the real world and incorporate movement, device and environmental conditions

and user scenarios.

Much traditional testing focuses on functional testing, and doesn't spend time on areas where the technology differs from traditional web and PC apps. On mobile projects, try to work towards what I call "barely enough documentation", or just enough documentation so that you can keep up and not get buried by maintenance in a fast-changing environment. Also be sure that the documentation overhead does not discourage high value testing in areas not covered in the requirements docs. If you look at I SLICED UP FUN, functional is only one of twelve possible areas to investigate for mobile projects. It's no wonder then that traditional QA departments struggle with testing mobile technology effectively unless they significantly change their approach.

Each project will be different, so be careful and use your judgment. I've even managed to strike a balance on fast-moving mobile projects in regulated environments with a large documentation burden. it just meant we needed to get creative and use technology to help with the extra constraints. There can be good constraints and poor constraints on projects though. A great constraint is the use of coverage models.

Coverage and Guidance

Cem Kaner defines coverage as "the amount of testing done of a certain type." (See the fantastic BBST course for

more on Black Box Software Testing[22].) We can expand that to the amount of effort when we are testing software using different approaches or techniques. We can discover different information if we test an application in different areas, in different ways, or with different techniques and with different tools. We might look at the same software, or even the same feature over and over in different ways and learn completely different things, just by using different test coverage approaches to change our perspective.

Coverage Models

Kaner describes a "model" as a representation of something we are trying to understand. Exploratory Testing[23] Software, hardware and related systems are complicated. As testers, we are trying to understand and learn more about them as we test them. Being able to focus testing coverage, and to report progress to stakeholders is a powerful tool. Note that I use a plural when I talk about testing coverage.

Coverage is often reported as a singular thing. Test case management systems generate metrics such as the percentage of tests that have been executed, and numbers of tests that have passed and failed. Programmers use code coverage tools to measure how much of their program code is exercised by automated unit tests that they have created. Both of these tool types tend to describe coverage as a singular, unified thing. This isn't correct. In his 1995 paper

[22] http://www.testingeducation.org/BBST/foundations/
[23] http://www.kaner.com/pdfs/ETatQAI.pdf

Software Negligence and Testing Coverage[24], Cem Kaner identified 101 different models of coverage. 101! I can think of a lot more than on that list. It can be tough at first, because we think of coverage in a narrow, singular way. Once your brain makes the shift, it is amazing what you can discover.

We can determine coverage through multiple approaches, and I have provided you with a few already. The tours chapter describes a good deal of approaches, and I SLICED UP FUN provides you with twelve more possible models. Add in techniques such as installation testing, upgrade testing, security testing, performance testing, privacy testing, usability testing, localization testing, load testing, failure recovery testingâ€¦ you get the idea I'm sure. You can (and should) define coverage in multiple ways. What I've provided here in the book is just to get you started. Please add your own to be even more effective.

If you have multiple models of coverage, you break down what often feels like a singular monolith of testing. It can be easier to prioritize and measure progress. You can focus in different areas depending on priority and need, and what will provide high value results the quickest. You can stretch out testing in each those areas as long as your schedule allows for, or if you are on a project with a tight timeline, you can get a lot of different kinds of coverage in short bursts.

Mobile testing leader Johan Hoberg uses attributes from the

[24] http://www.kaner.com/pdfs/negligence_and_testing_coverage.pdf

ISO/ISEC 9126 to help structure test approaches with teams he leads. The 9126 Quality Model has several attributes that teams can use to create models of coverage:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Each of these categories also contain several sub-categories of further coverage models. I encourage you to check out the ISO/ISEC 9126 (and similar models) for yourself, and see if it helps you generate more ideas to *create your OWN approach* that best suits *your* mobile testing project. You may find combining some of these formal approaches with what I have provided in this book will work well as a starting place.

Test Guidance

It can be absolutely fine to just charge ahead and test without planning, figuring things out as you go along. That can be fruitful to a point, but when we need things to be thorough, when you are managing others, and to have testers work in areas they may not be as familiar with, it helps to have direction. Furthermore, guidance documents

and systems can help us track testing efforts, status and progression.

Guidance documentation can help as a reference: "How do I do _____?" demonstrate common usage, and provide specific areas for focusing our testing efforts.

There are often two kinds of testers: those who wing it and feel comfortable figuring it out as they go along, and there are those who worry that they may not spend their time effectively without structure and guidance. The first group are often over confident, and the second group overly timid. Both groups will miss important problems as a result. The overly confident testers usually squander their time because they are too superficial in their approach, while the nervous testers waste energy dealing with their fear instead of focusing that brain power towards testing. Lightweight or "just enough" guidance helps reign in over confidence, and builds confidence in people who need it. Guidance that doesn't empower testers, doesn't answer some of their questions, doesn't help them do their work more effectively or bores them to death isn't useful. In fact, it is harmful because it steals time and energy away from value creating activities.

Guidance for Testing on the Move

Mobile devices by their very nature will require mobility when testing. You'll need to move around, and move the devices around to test your app's integration with location

services (it will be useless to do all that testing in one place), wireless internet access, movement and sensors, and different lighting.

How do you keep yourself on track if you go out for a walk or head off to another location for testing? What if you get distracted and squander your time? What if you are managing other testers? How will you know what they tested while they were away? I use some lightweight forms of guidance to help out. What's great about lightweight guidance documents is they are easier to maintain, they are specific enough to focus testing, but allow for variation, individuality and creativity, and you can compress time to work through them quickly, or extend time and work on them in greater detail if you have it.

Here are some examples of lightweight guidance documentation I have had success with on mobile projects.

Checklists and Coverage Maps

Coverage *models* are abstract, thinking tools, often used in when we are planning and strategizing our testing work. Coverage *outlines* are concrete, tangible tools we use as we execute and record our work.

My coverage outlines map to each of the coverage models we have determined to use on our testing project. (For example, we may have identified that we will use I, S, L, I, C, U, F, U, N of I SLICED UP FUN, as well as performance, security and error handing as the most useful

coverage models for our project.) While a coverage model is potentially limitless in scope, a coverage outline helps focus testing into what we actually think we can complete. Two of my favorites for mobile projects are checklists and coverage maps. Using the abstract models I identified above, I will then create short documentation to describe how we will implement testing in each of those areas. This is what we will actually use when we test to generate ideas.

Checklists

I create test checklists to help focus and track testing effort for each model of coverage that I need to have guidance for. If I'm testing on my own, I may have several models committed to memory, but for areas that I am less familiar with, or when I absolutely need to record results, I will have a checklist. I like lists, because I'm a word (or perhaps even wordy) person. I can create them in spreadsheeting software, or in a ToDo app or other productivity and tracking apps.

Checklists should be simple and easy to print out and take with you, or update on a mobile device. They contain a list of testing ideas that help me guide your efforts in a particular way.

Mobile experiences are highly personal, so coverage outlines such as checklists need to be specific enough to focus testing in a particular area, but vague enough to allow for personal interpretation. The *experience* is vital, and that is influenced by a lot of factors: the emotional state of the

person using the software, the environmental and wireless conditions, and others. It would be too much to try to predict all of these in our checklists, so I encourage each tester to observe the conditions around them as they test, and incorporate good, poor or mediocre conditions into their testing.

Here's a checklist story for you.

Years ago, I was involved with a project developing the first web-based bug-tracking software I had ever seen. I'm not sure if we were the first, but we couldn't find any other competitors at the time. We were an e-business company trying to cash in on the rise of the internet in the mid-late '90s, and we decided to use the web to help us be more productive as a company. We had a lot of clever design ideas, and incorporated the bug-tracking system into the project management/productivity app that we were developing. We also developed a test case management system. The design was full of great ideas, and I was very pleased when we rolled it out.

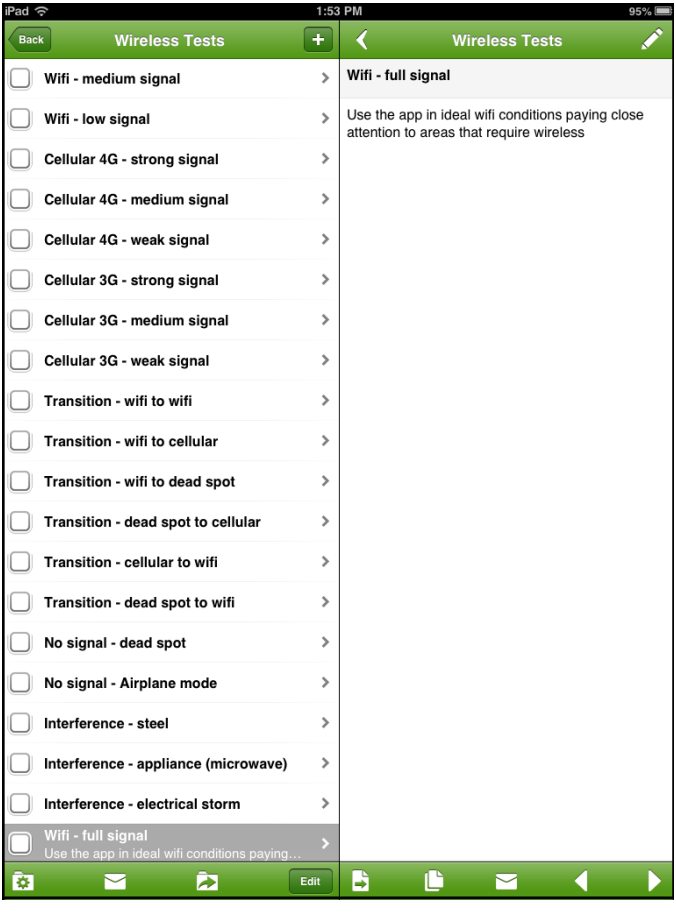
Unfortunately, our productivity immediately dropped, and our team started to miss important bugs that our customers were finding instead. In the past, our work had almost eliminated this kind of thing. Sadly, I realized that none of my testers were able to use it properly. While it met our requirements and had a lot of cool features, it slowed them down, and confused them. I spent time with each tester, and I found out that they all needed simpler forms of guidance.

They were new to the industry and I had trained them

well, but they lacked confidence on how to decide where to spend their time. To address this problem, I looked at their common activities, their goals, and their tasks. It turned out that while our system met their requirements superficially, it didn't gel with their work style, and was slowing them down. I came up with a simple solution that was incredibly useful: coverage outlines on spreadsheets.

Much to my chagrin, the cheap spreadsheets and a status board on a giant whiteboard in the hallway worked much better than the super cool testing productivity system I had co-designed and co-developed. It was disappointing, but I was pragmatic and went with what worked, so we stuck to the checklists.

This was a huge lesson for me. When you need creativity, valuable testing and speed in testing, use lightweight tools that encourage what people already do well. Don't impose a structure because you think it is ideal, or someone else told you to. Make sure it is suited to your people, your project and the technology.



Example Test Checklist

Coverage Maps

I like words and lists, but other people relate better to shapes and pictures. Coverage maps are similar to checklists in that they are a physical representation of a coverage model. Instead of a list of words, you use shapes. Mind maps are one popular way to represent them, and there are a lot of tools and apps that you can create and edit mind maps on.

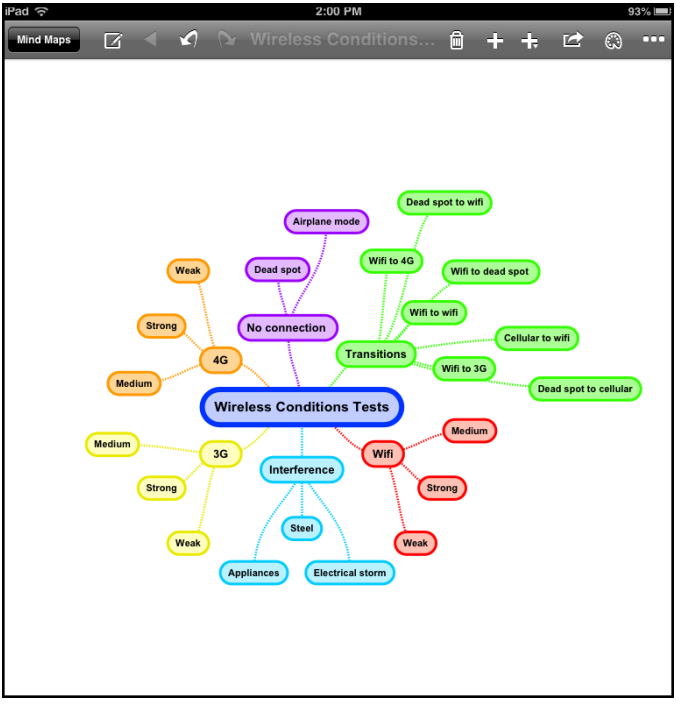
One of the first mobile apps testing efforts I led was made up of a team of people who were much younger than me. (Yeah, yeah, I know that after 15 years in the software industry I'm old. Now get off my lawn!) I noticed that the other testers didn't like the checklists as much. We were under extreme time pressures to get adequate testing coverage. We might get a build in the morning, and be expected to ship to beta customers by the end of the day. If we could find a show-stopper, important bug, then that would delay the release, probably by about a day. The programmers would scramble to fix it, and we'd scramble to test it.

Thankfully, the app was fairly simple, and we could get a lot of traction with testing in various ways by splitting up duties. We used three categories of coverage models based on application usage and criticality. We looked at primary usage, which were all features that were critical to the app being useful. If any of those features weren't working properly, the app was useless and couldn't ship. Next, we looked at supporting features. They were important, but the

app could still function if they were broken. Finally, we looked at optional features, or "nice to haves" that we had in the app. The app would be almost fully functional and our users could still get the job done if any of these features were broken, but it could be a bit annoying.

We broke testing up in this fashion so that releases could be rapidly deployed when needed, in the event that it was more important to have a mostly working app than a fully functional app. (There are competitive reasons for this, and in some industries at certain times they are a survival strategy.) We listed each of these features in a document, then made mind maps for each area of coverage. Each area had a test lead assigned to manage testing work. We could spend a few days testing each area, or we could spend an hour or two for a very shallow sweep.

Testers recorded results on these coverage maps either using mindmapping software, or by printing out copies and drawing on the paper. If we had a rapid testing session to determine whether an app could ship or not, testers would bring their sheets to a quick standup meeting with stakeholders and they would explain how much they had tested, and what features were working or if they had encountered broken or missing functionality. It was highly effective in a rapidly changing environment.



Example Test Mindmap

What? No Test Cases?

Test cases aren't very mobile friendly. They are cumbersome and hard to follow if you moving around. They take a lot of time to develop and even more to maintain because the technology changes quickly. They also tend to discourage variation and make it difficult to change test focus quickly. In many cases people *think* they need them, but alternative methods of keeping track and guiding your work can be just as effective to satisfy stakeholders.

How-To Guides

I get asked this question frequently:

"Without test cases, how do people learn how to test in areas they aren't familiar with?"

It turns out that test cases aren't a great way of describing that information. They tend to be specific and detailed and it can be difficult to understand how or why a feature works the way it does. To address the issue of people requiring certain information to do their work, I studied other disciplines: technical communications and instructional design. Based on that work, I create documents that are influenced by training documentation. For testing projects that require documentation, I create or re-use the following short docs:

- How-To Guides

- Testing Guides
- Calibration/Setup Guides
- Test References

If you are working with a larger organization, they may have a technical communications team who write how-to guides for products. Whenever I can, I work with the technical writers and get access to the docs they have written so we can utilize them as testers. I also get their document templates so we can follow their lead on style and format for the docs we create.

Testing guides can help direct testing, answer questions or generate test ideas. Rather than a generic how-to, these are short docs we as testers create to help others get up to speed on how to test something. For mobile projects, I supplement these with short video demonstrations. Video is easy to create on mobile devices, and we can quickly re-record them when the app or technology changes. It is much easier to re-record a short video than to edit a lot of words and screenshots in a long document.

Calibration or setup guides are for things that actually do require exact steps. I got this idea when I was sitting in a courtroom watching traffic ticket proceedings. (I was only an observer! Don't jump to conclusions!) Each police officer who had issued a speeding ticket were asked very specific questions about how they had calibrated their machines.

If they neglected to calibrate a laser or radar speed clocking device, the ticket would be thrown out. With testing, there

are certain things we need to do and follow to the letter, where variation in actions are detrimental. This can include setting up new builds on devices, generating and using certain kinds of test data, or certain complex system tests.

I also like to create short test reference documents. These might be instructions on how to get and install a new build for an app, who people are on the project and what their contact info is, or simply just a place that tells you where things are, and where more information can be found. I also include test information on the web, testing cheatsheets, or collections of test ideas that we have found useful.

Session-Based Testing

Some testers (and managers) worry about making the most effective use of their time when they are on the move. As I've mentioned, I often run into two types of testers. The first type are over confident and test far too little in the time they have available (I have full coverage on everything!) The second group, the timid testers have trouble getting started and waste energy on fear (I haven't tested enough, or I might have missed something important!) Helping focus all types of testers in areas of productivity, and giving them tools to demonstrate what they discovered when testing helps enormously. A fabulous tool for helping focus testers and enabling them to describe what they have covered with testing is Session-Based Testing.

James and Jonathan Bach invented a style of testing where you record testing *as you perform it*. This is the oppo-

site from test cases which involve pre-scripting testing and following steps, then reporting on what was pre-recorded. They called it Session-Based Testing Management[25] or SBTM.

Session-Based Testing is an incredibly powerful approach to documenting testing because it supports dynamic testing, where testers can change their minds and their focus (within limits) based on what they observe in the app to work on high-value activities, but still create documentation for regulatory or other reasons. I've used session-based testing on a lot of projects, and I've found a lightweight implementation of their idea works especially well on mobile projects.

Here are the components I utilize on mobile projects:

1. Clear Goal (called a *Charter* in SBTM, sometimes called a "mission")
    - A clear focus for testing
2. Time Box
    - Uninterrupted testing time (5-60 mins works well on mobile projects)
3. Reviewable Result
    - Something that you can speak to to help bolster your report. On some projects, these can be saved as project artifacts.
4. Debrief

[25] http://www.satisfice.com/articles/sbtm.pdf

- Go over what you did in the testing session with someone else, and discuss findings what to do next, etc.

Session-based testing helps focus testing, it provides goals to keep testing on track. Time-boxed sessions help remind us to concentrate and reduce distractions, so we can spend time on high value activities. They also cut down the amount of time spent on low-value activities if a session turns out to be a bust. It was only an hour of lost time in that case, not a day or a week. The time box helps testers avoid getting off track for a long period of times.

The reviewable result helps people record and describe what they did in their testing session. What did they discover? What is missing or preventing them from effective testing? The debrief helps stakeholders understand what had been tested, but it also spreads information around the team, reducing the need for one person to focus on a particular area. It also speeds up communication on the team. New risks, developments or technology issues can become part of the shared knowledge of the group very quickly.

Here is an example:

As I write this section I'm also testing a friend's local services app. He wants me to test it on an older smartphone platform for an afternoon. I will break up my time into 4 sessions:

1. Search in a service-rich area

2. Search in a remote area
3. Test the app with different wifi speeds
4. Test the app with different cellular connections and network transitions

Let's take the first goal, or session charter: "search in a service-rich area". I will write that down either in a note on a device, or in my notebook, and head out on foot towards a service-rich area full of restaurants and stores near my home. As I walk, I will search, and verify the results based on what I already know about the neighborhood. I'll look for places in out of the way spots and see if the app suggests them, and I'll look for my favorites. I'll also want to see if it can expand my view and point out something I don't know about that might be useful for me to check out. As I try these various scenarios, I will briefly jot down what I tested, and what my observations are, both good and bad.

I may even bring a second device to record problems on video. If there are location issues, gesture problems, or awkward workflows, a video can demonstrate what is difficult to describe. They are incredibly easy to record on mobile devices, and to send to either my email address, or the programmer's.

Once I have completed my sessions, I will debrief with my developer friend. We'll either meet up for a quick coffee and speak face-to-face, chat on Skype, or I'll just email my findings. He doesn't really care about my reviewable result as a project artifact, he just wants to know what the issues

are both good and bad. I don't just give him bug reports or raise concerns, but I also tell him what I liked about the app, and areas I particularly enjoyed.

Relentless criticism can make the whole endeavour feel hopeless, and that information can be just as useful for him to market his app, as the bug reports are for fixing it.

At the opposite end of the spectrum, when I was managing testing for mobile medical apps, our session sheets were project artifacts that would be reviewed by auditors from various regulatory bodies. We would record them in notebooks, or on paper, and then go back to our desks and enter them as documents following a specific format in a document management system. The form of the work is similar, but there is more work at the end to make sure the documents are easily reviewable, and they fit in with the rest of the project auditing requirements.

Optimize Testing Through Analysis

Have you ever spent several hours of your time on a task, only to find out you were working on the wrong problem? I've had it happen to me. You get going, spend time testing, and then realize you were doing the wrong thing at the wrong time.

Here's an example: when I was working on medical apps for mobile devices, a couple of testers charged off on their own to test application performance. They came back after working in different environments over an afternoon and

each independently claimed that the performance was just fine. Trouble was, we had reports to the contrary from some of our beta testers who were testing at their workplaces, in the real world.

Since the testers were utilizing session-based testing, closer scrutiny during their debriefs revealed that they were reusing web performance specifications instead of mobile metrics, and they hadn't spent time with any of our medical professionals. It turns out that medical professionals use computing devices very differently than many other users. They need rapid, instantaneous response and a high degree of accuracy so that they can not only make accurate readings of what they see, but so they can do them quickly. They have an enormous time pressure, and in some cases will lose their jobs if they don't process enough medical studies over a period of time.

Sadly, casual use by two testers did not reflect the needs of our users, and that testing effort had little value. It had to be re-tested once they had a solid understanding of our users and their needs, and they were then able to help the programmers identify and fix problematic areas.

I've also witnessed testing on the wrong devices. In one case, someone assigned to help the test team decided to test on their own Android device, even though the web app was only supported on iOS devices because that was all that our customers used. I've also seen people heavily test an app with a device on cellular networks on an app that didn't require an internet or network connection.

A bit of analysis up front can save a lot of time and effort. It can help you focus your work in ways that are important to your customers, your team, and of course you, the mobile tester who provides great work.

As I mentioned before, Cem Kaner describes a model as: "A representation of something we are trying to understand." Creating models of what you are testing is a powerful activity that can help your testing be more efficient and effective. You don't need to get fancy, just spend time researching and record your findings for each project you are on. Some of my models are just lists of attributes, while on larger projects with a documentation requirements, I may create diagrams and supporting documents. We'll go through some common analysis work that I do to help jump-start my testing productivity. If I have ten minutes or ten days to complete this work, I'll do what I can in the time I have.

Models are fantastic tools to help you:

- Exploit limited resources/time
- Generate useful test ideas
- Learn more about the application and related systems system and known weak spots
- Kick-start your testing thinking

Examples of things you can model:

- System identification

- Device capabilities
- Device interaction (hardware, sensors, services)
- Typical usage
- Users and common usage scenarios
- Error conditions
- Communication and networks

Modelling the Users

It is incredibly important to find out about the people who will use your application. However:

DO NOT GUESS.

I repeat, ***do not guess*** and assume that you know what characteristics, goals and activities the users of this application will be like. Also don't assume that they will be like you.

Talk to them, and find out their attributes, their needs with the software, and their comfort with technology. It is important to understand their *motivations and fears*, and how this application relates to them. If you can't talk to them, find out about them from others. Do some research. What age group are users typically in? Where do they use the app? What problems does it help them solve?

To research, talk to people who are designing and developing the app, and find out their target market. e.g. "accountants", "female high school students", "automotive mechanics".

Now narrow it down to different users within the target market:

- What are their skill levels and specializations?
  - Investigate what their typical use of the application will look like.
- What are their tasks and goals?
- Where and under what conditions would they use the app?

Here is an example to get you started. I had a test manager for a municipal (city) government IT shop in one of my mobile testing courses. They had designed and developed a mobile application for maintenance workers who inspected water and sewer pipelines. They had tested it thoroughly in their office, but she was worried they might be missing something, so she came to my course. We worked through a simple user modelling activity with her, and the results astounded her.

We didn't know much, but we were able to come to some reasonable conclusions based on her information, and the experiences of some of the others in the class who had worked similar jobs when they were students.

Here is what we came up with:

Q: What are their skill levels and specializations?

A: Trades people, a range of skill levels from unskilled laborers to journeymen to engineers.

Q: Are they comfortable with technology?

A: Some are, some are not at all. There is a vast age range, and younger people tend to be more mobile savvy.

Q: What is their typical use of the application?

A: Entering in data and readings from meters. They do this in vehicles, on foot, and in different locations.

Q: What are their tasks and goals?

A: They drive to monitoring stations and job sites around the city limits and enter in observations for alarm situations, maintenance needs, etc.

Q: Where and under what conditions would they use the app?

A: Anywhere, in any kind of weather, in tunnels (sanitation, subways and others), under poor lighting conditions, or in the dark. Temperatures drop so workers may be using gloves if it's cold. Some are mechanics, so they may have grease on their hands or wear protective gear.

We wrote these ideas questions on a flip board, and posted the answers around the room. Suddenly the manager had an "oh sh–!" moment. They hadn't been testing the app in typical ways it could be used by the sorts of people who would be depending on it to do their jobs. They'd been testing it the way QA people would, in a climate controlled office, sitting in comfortable chairs at desks, with fluorescent lighting. Oops. What about the engineer who is nearing retirement age, and struggles with new

technology? What about testing in different locations that might have different cellular connections, or weaker signal strengths? Wait a minute - what if they are underground inspecting different utilities? In those cases they are surrounded by reinforced concrete, which is full of steel. Will they even have a wireless signal in those cases? What happens if their hands are dirty? Hmmm… very good point.

Also, compare a mechanic's hand to my hands. You can't see them, so I'll describe them to you. I do a bit of work on cars here and there, and when I do my hands get stained with grease and oil and whatever else I rub up against. Most of the time though, they are very clean, look like a pianist's and the toughest surface they have to deal with is a keyboard, mouse and touchscreen on my mobile devices. Now think about a mechanic's hands. They have just fixed something, and their hands have grease on them. What happens if the touch sensor on the devices they support use capacitive sensing, which depends on the electrical activity that the human body gives off to work? If you guessed that it isn't going to work most of the time, you're correct! Now imagine that they have gloves on because it's cold. Again, the touchscreen gestures just aren't going to work if they are wearing gloves.

I had heard of something like this before. A consultant colleague of mine was asked to investigate a vehicle maintenance facility where a new touchscreen system had been implemented. Trouble was, the devices were getting destroyed in the shop. At first my colleague thought it was

because of the harsher conditions in a maintenance shop vs. an office. There is dust, dirt, grease, oil and other chemicals to contend with. My colleague arrived on site to find that the devices had been sabotaged. One looked like someone had taken a ball-peen hammer to the screen. It turned out that they were sabotaging them because they didn't work properly in their environment, and they were preventing them from getting their work done. No one had bothered to actually work with the end users, model their skills, activity, goals and common use, as well as the conditions they worked in. They took some requirements, and built something.

Also, based on her interactions with some of the workers, we created 3 personae, in the user experience style:

1. Jack, a 60 year old tradesman
    - Not tech savvy, gets impatient with technology. He has large hands and poor eyesight.
2. Travis, a 25 year old tech
    - Very tech savvy, forgiving of technology, but doesn't take care of equipment.
3. Les, a 45 year old engineer
    - Reasonably tech savvy, very patient, careful with equipment.

Imagine the conditions that we described above, where people are so frustrated, they resorted to destroying new equipment. Now outline the goals or tasks that they all

have to accomplish with the app. Now think of how each of the peronas will interact with the app differently. They will get frustrated by different things, and they will interact with the app completely differently, even though they all have the same goals and tasks. Touch and movement will be different as well.

The testing manager wanted to get out of my class right then and there and head back to her team to help them revamp their approach. They had found some good functional bugs that had been fixed already, but they were missing an entire universe of important testing. Those maintenance workers will never know, but if they did, I'm sure they would be grateful to that test team for thinking of them and focusing their testing.

Now as an exercise, get out a piece of paper, and try to answer those questions about a social app targeted at teenage high school girls. Don't guess though, talk to someone in that demographic. Talk to your daughter, sister, cousin, niece, or a friend of the family. Watch how they use the device, and find out where they use it, and under what conditions. Find out why they use it, and what apps they like and why, and apps they don't like and why. Then compare and contrast that with the questions we answered above.

This is an incredibly powerful modelling tool. Imagine if we tested the way we like, just in the test lab, but our conditions, usage and goals were far different? What do you think will happen if our end users are different from

us and we get it wrong? If you are thinking of a ball peen hammer smashing a device screen, you are thinking in the right way.

Modelling users helps us optimize our testing, and focus on high value activities. You don't have to be perfect at this, but try to be as accurate as you can. Any effort that builds an understanding of our users and helps us empathize with their usage of our apps is important.

Modelling Devices

It's important to understand what devices you will need to test, and those will reflect the devices your end users will most likely use. It's important to understand device capabilities and usage patterns so that you can test like a real user. Also, it helps save time if you can actually use the device when you start testing.

It's important to explore each test device type to understand how to quickly use device features like a regular user and to set up or create test conditions based on user actions and device states.

Here are some areas to explore on devices:

- User settings
  - What are different combinations of settings you can save?
- Communication integration

  - Can you generate phone calls, voicemail, text and other interruptions?
- Messaging: Information and Warnings
  - Can you generate interruptions from messages, warning messages or device states like low battery, or changing networks?

Here are some example user settings to learn about:

- Wireless and Networks
  - Different network types, data roaming
  - Prompt or no prompt to change networks/authenticate
- Location Services
  - Turn them off or on
- Sound, Display
  - Sound on/off, auto-rotate on/off
- Email, calendar, contacts
  - Enabled, disabled, auto update, manual updates, etc.
- Notifications
  - Permission to notify outside of app, turn on and off
- Language, keyboard
  - Non-English, different keyboards
- Accessibility
  - Voice control, reverse-high contrast display
- Date/Time, Location

  - Different time zones, locales
- Alerts and Messages
  - Communication: texts, email, instant messaging alerts
  - Calendar, reminders for appointments, alarm notifications
  - Connection messages: Moving to different wifi source
  - Low battery warning
  - Syncing to other devices
  - Style: banners vs. popups vs. others

I try to spend a day with a new device to explore it. This helps me develop the ability to use each device effectively for testing. I can now quickly generate actions and states on devices. If needed, I can also train others so they can get up to speed on the device quickly.

Different devices with different operating systems can behave very differently, and it can take time to get used to a new paradigm. It's also fun to try out new technology, so enjoy learning when you do.

Modelling Apps

Analyze the Product with TAP IT UP

One of the most powerful analysis tools that I have used in the past is the SFDPOT[26] (San Francisco Depot) mnemonic created by James Bach. It helps us look at an application from different perspectives to learn as much as we can before we start testing. Each letter in the mnemonic can also be used to generate test ideas. *S: structure*, or how the product is composed, *F: function*, what the product does, *D: data*, what the product processes, *P: platforms*, what the product needs to work, *O: operations*, how the product is used, and *T: Time*, any effect time has on the product. Recently, an I has been added: *I: Interfaces*, how the product can be interacted with. This is a powerful thinking tool, and if you would like to use it, open up your web browser and search for more information on it.

The first time I saw James test software and use this mnemonic in real time to rapidly change his testing perspective it blew my mind. He was able to find a wide variety of bugs in a very short period of time. (Problems I had missed.) Prior to working with him, I had only used it as an analysis tool, and I still do. In fact, it's one of the first thinking tools I reach for when I am planning to test any software. However, on mobile projects, I find that while it's useful, I needed to adapt it a bit.

[26] http://www.satisfice.com/articles/sfdpo.shtml

SFDPOT was developed for PC technology, and while it still works on other technology, I found I needed to customize it so it would be more useful testing mobile apps, supporting systems, and other sensor-based computing devices. After a while, it wasn't recognizable, so with the help of my wife Elizabeth, we came up with a new, similar mnemonic to help you analyze mobile products prior to testing them:

TAP IT UP

T: Tasks the user wants to complete with our app

A: Application composition and features

P: Platforms we will need to test

I: Inputs and Outputs, such as gestures, files, data feeds, or interacting with external systems and other devices

T: Time and Timing and its effect on testing

U: Usage and typical user interaction

P: Places where the app is used

Now let's expand this mnemonic and go into more detail on how to make it real on your project.

T: Tasks

What people will achieve with the app.

What are the primary tasks our users will use the app for, and what goals will they be trying to reach? They may want to answer a question, solve a problem, or be entertained.

To put context around the tasks, I use the mnemonic **TAG** which stands for Tasks, Activities, and Goals. This helps me think of motivations behind usage. For example, imagine a banking application. I may have a task to check a bank balance, which I am doing because a creditor called about a missed payment, and I may be doing this while I am watching TV, or standing in a parking lot outside. In less stressful situations, I may check out some of the features of the app and learn about different things I can do with it. I may use some of the features casually after I have reached my goals with the app.

A: App Composition

What the application is composed of.

(This is inspired by the "S" in SFDPOT.)

What mobile features and services does this app use? (eg. user interface design and components, gestures, sensors, location services, etc.) We talked about a lot of them in chapter 3, and it is important to understand how the app is composed so that we can focus our testing correctly. What mobile technology are we using? Is it a native app, written for that operating system, or is it a web app using HTML5 and CSS3? Or is it a hybrid app, using JavaScript and web plugins for the user interface, but accesses sensors through an API?

Are there external systems that the app interacts with? Do we have a web server that supplies it with information? How about 3rd party data feeds from other companies or

partners?

Here are some perspectives to consider. While you can figure some of these out by using the app and experimenting, you'll need to talk to designers and developers to get a thorough picture.

Application Technology:

- Frameworks, languages
- Mobile features
- Location services
- Social features
- Entertainment and gaming features
- Web features such as embedded web views

Internal Systems (non app systems required for use):

- Web and application servers
- System architecture, diagrams
- Networks, hardware and storage

External Systems:

- Advertising
- Content or information streams
- 3rd party relationships

One trick I use once I discover this information is to go to my favorite search engine on a web browser and search for "HTML5 sucks" or "problems with Android 2.3". (Plug in the technology your app uses of course, don't just copy these.) I will quickly search for known issues related to the composition of our application we are testing. This helps me to quickly discover what some of the problem areas are, and I can exploit those when testing. Want to blow a programmer's mind? Do this to inform your testing, and you will find out about problems that they didn't know existed.

P: Platforms to Test

What the application needs to work.

(This is inspired by the "P" in SFDPOT.)

This is a testing analysis perspective taken directly from Bach's San Francisco Depot mnemonic. We've talked about platforms before in the book. First off, you need to figure out what operating system your team is targeting, and what devices: smartphones only, tablets only, or both. From there, the platforms get complicated. Here are some examples to help you start thinking about what you will need to test.

Native Apps, No Network

- Hardware - smartphone
- Hardware - tablet
- Screen size

- Operating System version
- OS sub-version
- Locale (location and language)

Native Apps, Network

- Hardware - smartphone
- Hardware - tablet
- Screen size
- Operating System version
- OS sub-version
- Wifi
- Cellular networks
  - 4G, 3G, EDGE, others
- Carriers
- Locale

Web Apps

- Hardware - smartphone
- Hardware - tablet
- Screen size
- Operating System version
- OS sub-version
- Wifi
- Cellular networks
  - 4G, 3G, EDGE, others

- Carriers
- Locale
- Web browsers (most devices support several)

Add in transitions between networks, weaker network signals, secure communications and you will have even more combinations to consider.

I: Inputs and Outputs

What the application processes.

(*This is inspired by the "D" in SFDPOT.*)

Touch Screen Inputs

You can type and gesture to input various things into the app. However, watch for inadvertent actions as well, accidental presses or gestures when holding a device.

Off Screen Inputs

Many devices are now supporting gestures and movement for inputs that don't require touching the device, but by detecting movement near the device. This can be even more difficult to test, and more prone for error.

Movement

If an app uses movement sensors, find out which ones are utilized and what they are supposed to do. Find out what happens if you combine movement with other input activities. Also, location services and inputting or sending/uploading information while on the move is important to consider.

Voice

Does your app allow you to control it through voice control? Does it allow built-in voice control apps or services to launch it and interact with it? Can it record voice files to be replayed later?

Video and Sound

Does the app depend on input files or other files to function? What might happen if they are missing, or are too large, or corrupt?

Accessories

Can you use alternative inputs? What are they?

Plugging in a power or syncing cable causes different sorts of inputs: electricity to power and charge the battery, or updates when syncing with a PC or other device.

Many devices are going back to using stylus input devices to help with gesture inputs. There are also special gloves that can be used in colder weather that allow for touch screen interaction. Find out what your user community uses, and test with those accessories.

Files

Find out if the app uses files or a database to update information, and if it sends output files, or database queries when things change.

Information

Information can come from other websites, from datafeeds and other services. This can be quite complex, so it is a good idea to find out what each of them are, how much data they utilize (either to or from the app) and whether some of them occur at the same time.

Other Devices and Screens

We now use our mobile devices while we use other devices, such as a PC or a Television. We even co-ordinate or control other devices with our mobile devices. We might update a media center by downloading a movie or song from our device and saving it to something else, or we might use our device as a remote control to drive a television or PC.

Payment processing with wireless technology is also an area that involves inputs and outputs. We just have to hold our device close to a payment system in a store, and we can pay for goods and services with our mobile devices.

Syncing and sending files or other information can also be done using technologies like Bluetooth or RFID.

T: Time and Timing

What effect does time, or timing have on the app?

(This is from the "T" in SFDPOT.)

Time and timing are incredibly important to take into account when testing mobile apps. Find out when apps update with new information, and if there are multiple things occurring at the same time. Some favorites for developers are during app launch, or when you submit something after

typing. To make the app more efficient, several things tend to happen behind the scenes, but if any actions fail or take too long, they can cause failures. If you find out about how long it takes for an app to time out and go into an error recovery position, or areas in the app where several things happen at once, that can be incredibly useful when finding problems, or narrowing down intermittent bugs.

Time is an enormous factor when tracking down intermittence, and in many cases I have found repeatable cases for seemingly unrepeatable bugs by identifying things that were going on at the same time, only at certain times to cause intermittent errors.

U: Usage and User Interaction

Who uses the app, how they use it, why they use it.

This is one of the most important factors when analyzing an app. Find out who the users are, what they expect from the app, and how they use it. What are their motivations and fears? For example, in a banking application, my motivations might be to check to see what my bank balance is, or to make sure a payment was made on time. A fear would be a loss of money, or that an important transaction failed. Fear can be a huge factor in testing, because it can completely change our need for fast performance and we depend heavily on its ease of use. As I mentioned in the user modelling section, do not guess about this, but do some solid research to find out for sure.

P: Places Used

Where the app is used, and under what conditions.

Where an app is used, and the conditions surrounding it is important for mobile apps. Some locations have strong, clear network signals, location services work with ease, and they have controlled temperature and light sources. Other locations have completely different characteristics. For example, if an app is used in bright sunlight, it may get hot, and heat optimization may start on the device itself, slowing down performance and negatively impacting your app. Poor network connections, and other factors have an enormous impact on how apps work, and if they work at all. Understand where they are used, and emulate those conditions as you test. Do not assume they will be used in the often-ideal conditions of an office.

Modelling Entire Systems

An app might work just fine and dandy on its own, or in the ideal conditions we develop them in. However it might not work if the systems it depends on aren't working, or are performing poorly. Many apps and servers do not handle issues when they get out of sync with each other.

Just this morning, we had a repairman in to fix our clothes dryer. He had a tablet with a POS (point of sale) device that he used to manage all the paperwork and details of each service call. Since I work in the mobile industry, I asked him to show me his device and app, and asked what he liked and didn't like about his mobile system.

This question caused him to go on a 5 minute rant, while he demonstrated problems with poor performance when interacting with the app with gestures or a stylus. The worst problem was with slower cellular network connections. He would fill in all the customer information about the call, and send in a form to their servers. When he got back to the office at the end of the day, there would be a lot of errors in the service records on the server, even though they were fine on his device. He would then have to spend extra time updating each service record manually. He wondered why some of the information would go missing. My guess is that with a slower cellular connection there was packet loss, and the server didn't do any error checking on the network communication, and just filled in what it had received and was valid.

As a tester, I try to get a good picture of the systems that support our app, if they are required so that I can identify potential points of failure. Here are some questions that I ask:

- Does our app depend on an internal web server or other system that we control?
- Does our app depend on external systems?
- Does the app have social networking integration?
- What happens if the app doesn't get useful data, information or updates on time, or at all?
- What would happen if cellular networks have weak signal or are congested?

- Can our app and the servers recover if they get out of sync with each other?

Here are some areas of supporting infrastructure to research:

Internal:

- Web and app servers, storage devices
- Network systems and related technologies
- Hardware:
  - Physical servers, clusters, disks, routers, hubs, switches
- Software:
  - Frameworks, engines, operating systems

External:

- Telecommunications systems
- Cloud services
- Third party data and information providers

To support testing, I determine critical points of failure, and try to find out what could go wrong if any part of the system fails, or has poor performance.

Model the Market

Stay on top of developments so that your efforts don't get blind-sided by a new release, or a rapid change in technology. The mobile market moves at a fast pace, so don't get too caught up in the day-to-day details, but at least understand what is coming for the operating systems and frameworks that you are testing.

- What's popular?
- When is the next release for the OS or new devices we support scheduled?
- What are the new features and challenges to expect?
- What is on the decline?

This information can help us as we plan our testing. For example, if we know that a popular new version of a device we support has a major release half way through our project, we can plan for that accordingly. If devices that we support have dropped sharply in usage in the market, we can reduce our testing work on those devices, and focus more on popular devices.

Final Thoughts on Modelling

Don't get overwhelmed by this effort at first. It will take a while to learn about all this information, so pick it up gradually and incrementally. If you stay on top of it and set

aside small amounts of regular research time daily, you'll be surprised how much you learn in a relatively short time.

After you have a few releases under your belt, much of this information will be in your head, and you'll intuitively know how to focus some of your testing work without thinking too hard. Keep at it and be consistent - it's a lot of fun to learn and keep up with what is going on in the mobile world.

Test Planning

If you are working for a software company or in an IT department, you may be required to create a test plan. This might be a formal document, or a few points on a wiki page or story card. The document and the form of the document aren't as important as the contents. If it helps focus and direct testing towards high value activities, and helps you and your team make the best use of resources, it's a good plan. If it doesn't, then scrap it.

Creating a Test Plan

At the very least a test plan is a combination of your test strategy and logistics information. I learned that from James Bach. A strategy without logistics won't really get anywhere, and logistics without a focus can be easily wasted.

I finally learned how to plan well by reading James Bach's work. I never liked my test plans, and while researching on how to do them better, I stumbled across his articles and worksheets on planning. He just described taking your analysis work, your test strategy, and risk analysis work, and putting those together with your logistics, and presto! You have a plan! Simple, yet brilliant. I also learned from James that a test plan doesn't have to be some long, onerous document. If you aren't required to create a test plan document, a plan can be in your head, or a few scratches in a notebook or on a whiteboard. On mobile projects, it could even be a list in your favorite ToDo app. The plan starts in your head, and it is a dynamic, changing thing. When conditions change, the plan must change as well. It also should be useful, not just something that is filled out because people expect it, but never read it.

On some mobile app development projects, my test plans have taken various forms: a short overview on a wiki page, overview on a whiteboard, areas and tasks written on post-it notes or 3x5 recipe cards on a status board, and in regulated environments, a formal document that met required specifications and formatting for auditors. Each was successful.

When I plan, I use a combination of analysis (product and risks), a strategy (a roadmap for meeting an objective) and logistics information. I do this for each project - they are custom made, based on research and have a goal in mind. They are not a template downloaded from the web, or

copied over from the last release. They are also never 45 pages long. They are to the point, as short as possible, and I do what I can to make them enticing. They are at the very least useful for me, and I refer to them frequently over the course of a release, and update them as changes inevitably arise.

A test plan is NOT a template you downloaded or borrowed and filled in. If you do this, you will miss important areas and perspectives to test, and I will be very cross with you!

Here are the areas I work on to create a plan, whether it is in my head, or whether it is a formal, audit-able document. To create your own plans, I highly encourage you to check out James Bach's work on test planning.

While these references are very dated, they still provide a great foundation for helping start test planning:

- Heuristic Test Planning: Context Model[27]
- Heuristic Test Strategy Model[28]
- Satisfice Test Planning Guide: Building the Plan[29]
- Test Plan Evaluation Model[30]

[27] http://www.satisfice.com/tools/satisfice-cm.pdf
[28] http://www.satisfice.com/tools/htsm.pdf
[29] http://www.satisfice.com/tools/build-the-plan.pdf
[30] http://www.satisfice.com/tools/tpe-model.pdf

Planning: Analyze the Product and the Project

Analysis helps us to determine what to test, and what areas we can ignore, or spend less time on. We can also use our analysis work to understand the problem space from different perspectives. So how do we do it? Well, I've given you a lot in this chapter already:

- Determine if you have any information on coverage and guidance. If there is none, find out what can help you determine coverage and guidance for this project.
- Modellingthe app, the users, typical use, devices, supporting infrastructure, the conditions where an app will be used

I've given you some ideas to help get you started, but this is an area where you will have to really use your brain and your judgment. Also, different projects have different opportunities and constraints. Some projects will have more opportunity for thorough work, while others will be rushed. Try to spend a bit of time on this though, even in extreme cases, because it can help you focus your testing towards high-value activities.

As you practice and learn more, you will also be able to incorporate prior work into new projects.

Applying Analysis

Once you have done your research, there are several things you can do:

- Perform assessments to better understand the testing project
- Create coverage outlines
- Map out your models to improve understanding and spot potential weak spots
- Execute tests to learn more, and revise as necessary

Planning: Assess Project Risk and *Rewards*

Risks

One of the most common methods for optimizing testing work is to use a risk-based testing approach. A risk assessment is a great way to start: you and your team identify threats to the project's success, and potential weak areas with corresponding testing activities to help mitigate the risks. I like to look at these areas and determine threats and weaknesses to project success:

- App requirements and features
- Business needs (sales, marketing, etc.)
- Technical issues or limitations
- Usability factors

- Legal, organizational and environmental considerations

I start of with the basics. I enumerate the core features of the application, and what would happen to the product if they fail to work correctly. This is quite obvious, and often overlooked, but it is an important consideration for testing. I then move on to sales and marketing, and what their values are. If the app doesn't work, it will be hard to sell. It's also really important to understand what the people in charge of sales and marketing worry about. If you understand their fears, and what keeps them up at night regarding the success of the product, that will help you understand risks much better. You can even test to help mitigate the risks that they bring up. (As long as they are rational and don't involve zombies.)

From there, I talk to programmers and designers and ask them about their fears. What makes them nervous from a technical perspective? Are there new features or tools that aren't as stable as they would like them to be? What have they introduced into the app that keeps them awake at night? How can we mitigate those risks through testing?

On mobile projects especially, I also list out usability concerns as a risk. This is incredibly important, as we've discussed in this book already.

I also look at organizational and project requirements. Do we have any requirements from regulatory bodies to meet? How about legislation and local laws? For example, if we

store customer information, that needs to be kept private in accordance with local laws. If there is private information, that needs to be stored securely. If your app is distributed by a mobile carrier, they may have oversight requirements as well.

I start out listing something like this:

Risk: The search function fails.

Potential Result: The app will be inoperable.

Mitigation through Testing: functional testing, usability testing, related tours

Risk: The app is difficult to use.

Potential Result: People will rate our app poorly, and may even rant on social media hurting our image and sales.

Mitigation through Testing: usability testing, performance testing, related tours

Technical Risk: We are using a new search library.

Potential Result: This is a new version with less field testing, so it may have some side effects.

Mitigation through Testing: thorough functional, performance and error recovery testing

I work my way through each risk in this fashion, and once I have a list, I share it with other teammates and see if I have missed something, or got something wrong. Once the team is happy with the risk assessment, I can now use it to help narrow down our testing efforts.

When I am working in regulated environments, mission-critical apps, or on security projects, this is much more thorough because it is required.

Rewards

Risk-based testing is quite common, and a lot of people have written about it. However, the negative side of things, the weaknesses and the threats, are only one side of the picture. On the other, there are opportunities and strengths. One development manager asked me to help his testing team because they were used to risk-based testing only, and were holding up some testing projects.

It turned out that this company provided mission-critical software in regulated environments, but they were branching out to explore other products. When a prototype had been developed, they would demonstrate it at trade shows or on sales calls. They wanted the app to be good enough for a demo, and if they got enough interest, they would then decide whether to proceed with it or not. Trouble was, the testers were holding up releases on these demoes because they wanted to do a risk-based approach. They were nervous about testing something lightly, even though the risks were not related to malfunction, but to a poor demo or poor product idea. They had their priorities for testing backwards.

I tried to figure out a different approach to help them. I went for lunch with my friend David McFadzean, and asked him what he thought. I explained the risk-based

testing approach, and asked him what he thought the opposite would be. He gave me a side-long glance and said: "rewards. Rewards are an opposite of risks." AHA! That was perfect.

I went back to the team and shared this insight:

- Risk-based testing focus is find out where the bugs *are*. ie. where did we fail?
- Rewards-based testing focus is to determine where the bugs *aren't*. ie. what did we do well?

I then created a rewards-based assessment. I shared that with the team, and they were able to change their perspective and test more appropriately, and in much shorter periods of time. Instead of trying to find all the bugs we possibly could, we would go over sales demo scripts and make sure that everything they wanted to show worked as expected. Bugs were logged if a sales demo script failed in a critical area. Testers would also figure out workarounds for some trouble spots. The point of the release was to have a successful demo, and this testing was focused towards that.

Looking at rewards, or positive testing is great for prototypes used in sales demos, or when teams are encouraging new development and need to win over stakeholders. However, it is also incredibly useful to look at when we are testing any app, because it helps ground us. It balances the negativity that we often face and even project when testing with something positive. That can be a huge encouragement for our team mates (Cool, this new feature rocks!),

but it can also be a powerful perspective change for us to find different issues.

Planning: Test Strategy

I have a whole chapter devoted to strategy, because it is an important factor to consider when planning. Great testing projects that have amazing results tend to have good strategies. A strategy bridges an abstract idea like "go test this" to something concrete. A good strategy guides and focuses our test execution, and maps our testing work to our goals for testing on a project.

As a consultant, I am sometimes brought in to lead testing projects that are in trouble. In some cases, the testers are quite low skilled, or inexperienced. Sometimes, the testing group I have to lead is incredibly dysfunctional for a variety of reasons. One of my tricks to get high-value testing results is in my test strategies. A good strategy will take various factors and limitations into account, and there are a lot of problems that you can work around if you focus your testing work, play to people's strengths, and use it to help your team do better than they did last time.

Analysis and risk/rewards assessment work inform my strategy. Solid analysis and a good understanding of what risks to mitigate against, and what rewards to exploit to make our product awesome will help me get a great strategy together.

When I am planning a testing project, my strategy is the

heart of the plan.

Planning: Logistics

Logistics involve anything that relates to tools, people, budgets, deadlines, schedules, rooms, resources and whatever else we need to actually implement our strategy. As Bach says, putting together strategy and logistics makes our plan. The two depend heavily on each other for success. Just look at the military. No campaign will win unless the troops, equipment and supplies arrive at just the right time. If you drop in troops to an enemy zone, but you can't supply them with ammunition and food, they are not going to be able to sustain an attack. If a battle or skirmish requires air support, but the aircraft drop bombs at the wrong time, they may miss enemy targets, or worse, kill their own troops. The history of warfare is littered with failures that were due to logistics problems.

Antoine-Henri Jomini was an influential French General who wrote extensively about military strategy. He said: *"Logistics comprises the means and arrangements which work out the plans of strategy and tactics. Strategy decides where to act; logistics brings the troops to this point."*

The same is true of our testing projects. Everything else can sound great on paper, but if we don't have the budget for devices, or people available at the right times, or even the space to do our work, then our plan will fail.

Anything related to the tools, hours, availability that is re-

quired for us to succeed is taken into account. For example, if we need to get new software licenses on PCs, or update or purchase new devices, we need to account for that effort and the timing to minimize downtime.

On a smaller project, I may be a team of one. That puts a lot of strain on my resources for testing. I don't have a lot of tools, and I don't have a lot of devices. I also only have a few hours in a day to do the work, with no one to help me. In this case, I may utilize some sort of verification service using crowdsourcing or remote device access services to help.

On a larger project, I may have to balance my needs across the rest of the organization. Key people may be committed to other projects, and I may also need to book time for test labs, training and knowledge transfer, and other activities. On a large project in a larger organization, a lot of my test lead work is in project management. I have to walk a fine line between my budget, people's availability, and getting just the right thing tested at the right time. It's difficult to co-ordinate.

That said, I have been able to out-perform much larger teams because of our superiority in strategy development and logistics management. I once led a team of three testers (including me) on a mobile device project. We were given a week to test to determine whether an intermittent bug was occurring or not. I spent a solid day on strategy and planning while the other two testers supported me with their testing activities on the device.

Once we had a game plan, we carefully orchestrated a week's worth of testing using various perspectives and coverage models based on risks and project constraints. Another testing team of 8 were located in another country working on behalf of a partner company. They copied and pasted their test plan from the last release and did some find/replace and ran through their regression test cases in about one day. We were under enormous pressure to cut our testing time short, but we found showstopper bugs that they completely missed due to a more well thought out approach.

This happened several times. The two companies were so impressed with our results that they flew the QA Manager from the other team all the way from Europe to North America to find out what we were doing differently. They were absolutely amazed at how we were able to do much higher value testing with fewer people.

That wasn't because we were smart and talented and skilled (ok, that was part of it) but it was mostly due to our strategy and logistics work. It's a bit extra work to spell out and do, but as you get more practice and knowledge it doesn't take that long.

Reporting

I also like to include reporting into my logistics planning. How, when and what I will report to other stakeholders is noted as well. I try to find out when they would like updates, and it what format. For the testers, I will use lightweight tools like wikis or status boards to track bugs,

the severity of bugs, and what our testing coverage looks like. This information is important for stakeholders as they make decisions about the product. Here are some example reporting tactics I may use on a project:

- Frequent debriefs with colleagues to describe what was tested, especially when using session-based test management
- Visible bug reports, coverage status and metrics on bug counts
- Big visible charts to show the rest of the team what our progress is with regards to bugs and coverage
- High level reports to management in documents, emails or verbal reports (it depends on the organization)

Quality Criteria

Another crucial aspect to test strategy and planning that I learned from James Bach is the use of quality criteria. I am often asked: "How will we know if we're ready to ship?"

A lot of teams like to use bug metrics, and that is one aspect that can be used. If there are no such and such severity bugs, then we can ship. However, that doesn't necessarily cover areas that are important to non-technical people on the project. More sophisticated projects will look at the following when making a ship or no-ship decision:

- App is feature complete

- Bug count is as close to zero as possible
- Testing coverage is thorough enough
- Regulatory or other requirements are met

In addition to these sorts of measures, I like to develop quality criteria by determining what stakeholders expect the software to be like. Often, non-technical people have trouble providing specific ideas outside of: "we want it to be awesome and kick ass!" TO help with that, I provide a base group of criteria that I got from HP called FURPS[31]. The software should be:

- Functional
- Usable
- Reliable
- Perform well
- Supportable

This is a good place to start. With the functional criteria, we want to make sure it has the right set of features and attributes. "Usable" is obvious, but it should also be usable to *them*, so they need to use it, not just watch us demonstrate. It should have good *performance*, and it should be straight forward enough so that we can support it if there are problems encountered in the field.

I often tweak this, I add in an extra P for "polished" which is one that my old friend Javan Gargus suggested. *Polish*

[31] http://en.wikipedia.org/wiki/Furps

means that the app looks right, that it is professional and holds its own in the market. I also often replace "Supportable" with "Secure", and we demonstrate that to stakeholders through our development and testing mitigation strategies. Sometimes the other stakeholders agree with what I provide, and that's the end of it. Other times, it sparks their thinking and they supply other criteria that are important to them. As long as the criteria are reasonable and measurable in some way, they will work just fine.

The key with quality criteria is they get information and impressions from non-technical people that the rest of us might have missed. I like to use surveys based around the quality criteria to help us get that crucial information that may be difficult for non-technical people to express to us. I have had all of our other measures come up positively: no high impact bugs, great test coverage, feature sets complete, designers, programmers and testers are happy, but the quality criteria survey was failed by someone who noticed something the rest of us had missed. They are valuable tools to incorporate into our release decisions. What's more, they get people involved into the project who might feel left out during the technical work, and they feel like they are still part of the team.

Final Thoughts on Planning

When I'm finished the work I have described above, I have four or five short documents:

- 1-4 page analysis doc, sometimes with related model diagrams
- 1 page logistics document with tools, devices, people, schedules, deadlines, etc.
- 1-2 page risk/reward assessment spreadsheet
- 1-4 page strategy document (if there are a lot of device considerations, etc. it can go over one page)

From there, I can create a test plan of whatever length it needs to be. If the plan needs to change, I still have the working docs, so I can file away the old test plan, update the working docs, and crank out a new one. It's relatively painless if we shift mid-project, and on mobile projects this can happen a lot.

Test plans have a long history on software teams - a lot of effort is put in to create them, but most of the time, no one reads them.

When I was a newbie tester working for a software company, I was excited when I was put in charge of a testing project. As a test lead, one of my jobs was to co-ordinate the efforts of other testers, but a big part of my job was to create and maintain a test plan. I was given a template to fill in, and I spent days analyzing new features, figuring out what sorts of tests we were going to run, updating logistics information, and making sure I followed the documentation standards of the company. When it was time to get my test plan reviewed, I felt a sense of accomplishment. I sent it out to other stakeholders and asked for their feedback,

and then I waited. And I waited some more. I finally went around in person and asked if anyone had any feedback, and I got a lot of hand waving and "I haven't had time but I'll look at it later" responses. Then it was time to test, and I was supposed to have approvals and feedback, but we just forged ahead. Eventually, people just nodded their assent when I asked if they were ok with my test plan. I didn't think they read it, but every release as a test lead, I had to take the template, fill it in, and go through the process again.

Eventually, I started looking into *why* people weren't reading my test plans. First, I started with me. What did I like about them, and what did I dislike? Mostly, I didn't like that we had templates that we filled in, which were usually the plan from the last release, or a similar project with details changed. For example, the test plan for the release we were working on now would be identical to the previous release, except for dates, names and version numbers. By identical, I mean we copied the plan, and made changes. We would delete the new features from last release, and replace that section with the updates, and we might have different people involved, but other than that, it was identical. What use did that provide? In fact, I didn't really use the plan other than to refer people to if there was a question about logistics, or after the fact if there was a problem. So I started my next plan from scratch - it would contain information that would help me as a test lead, because if no one else was reading it, I may as well make it useful for me and my team.

I made a few attempts at test plans, but I didn't really create anything that met my needs until I found James Bach's Test Plan Evaluation Model online. He had this great document that he used himself to evaluate test plans that he had made publicly available. I read it, and also found he had a document describing how to create a plan, called the Test Plan Building Process. I noticed two things right away. First, his test plan building process had four components: product analysis, risk analysis, test strategy and planning logistics. Put those four components together, and you have a test plan. The second ting I noticed was this document was in the form of a checklist. It was short, easy to consume, was laid out attractively, it was lightweight, but packed with information. It was 3 pages, and it was incredibly helpful for me to think about how to focus and organize my testing work. I had created 45 page test plans that weren't nearly as useful as this.

So I went to work to answer some of the questions raised in this model. I spent time working on understanding the product, the market and our end users. I asked about risks and areas that made different stakeholders nervous. What did they fear might happen if we got something wrong in the release? Conversely, what did they think success looked like? I asked programmers about areas of concern, as well as designers, customer support, managers and technical writers. From that work, I came up with a list of risks, or threats to the success of the project, and then I recommended testing techniques, approaches and tools to mitigate those risks. Then I looked into logistics. Who was

available, and when? What equipment did we need, and when did we need it? Were we missing anything that was required for our project? After a few days I had completed this work, and I had four short documents. I started asking people for feedback on each of the areas, rather than giving them a test plan document. I made some changes here and there, and worked on my objective for the test strategy. I felt the strategy would be the backbone of my test plan, and I wanted to clearly state an objective for testing so we could see whether we met it or not at project end, and I wanted to state how we were going to reach this objective.

At the time, I was in a regulated environment, so we had to provide a test plan, and there was a good deal of pomp and ceremony associated with it. There were signoffs, regulatory-required cover pages, headers and footers, and proper folders for archival and auditing purposes. This was an important test plan, but I also wanted team buy-in and feedback. I had watched the lead programmer on my team for several weeks, and every few days a project manager would hand him some sort of documentation, and he would wordlessly toss it in his bottom desk drawer. It was a bit comical, I'll try to describe it for you. We sat in an open office environment, and we sat across from me with his back to me. There was a very short partition, so we could see each other if he turned around. My usual view of him was the top of his head poking above his huge comfy office chair, and occasionally an arm or leg would stick out. So when a project manager wandered up and handed him some documentation, the only movement I would

see was an arm reaching out from the back of the chair, grabbing the paper, opening the desk drawer and dropping the papers in. His head did not move at all. Around the time I was ready to have my test plan be evaluated by him and other team members, I popped over to his desk after I had watched yet more paper go into the bottom desk drawer.

I asked him what that desk drawer was all about.

> "What do you mean?"
>
> "I just see you put papers in there that people give you."
>
> "OH!" He started laughing, and opened up the drawer to show me.

> "That's where project plans go to die." I thumbed my way through the papers. There was about 8 inches worth, all in geological layers, going back several years. I was determined that my plan wouldn't go into that drawer. Maybe he could provide some insight? "Why don't you read them?"
>
> "Because they are pointless, and have little to do with what we are actually doing."
>
> "But I have to create a test plan for our project."

> He started to wrinkle up his face.
>
> "Can you give me some feedback on it as I develop it so that it is at least slightly more useful than the documents in the drawer?"
>
> "I guess so, if you have to. It would be better to have A>something that isn't completely lame."

After that exchange, I worked on my strategy so that it was clear, actionable, short, and to the point. Once I was happy with it, and the testers I was directing were ok with it, I started filling in the test plan. I used the documentation style guide, and began with an opening paragraph describing what the product was for, who the users were, and what the organization hoped to gain from this release.

I put in my test strategy, with our objective, and how we were going to try to reach that objective. To bolster the strategy section, I added my risk assessment information, and added that in, highlighting the largest threats to our success, and what testing we were going to undertake to mitigate those risks.

Finally, I added logistics information: who would do the testing, where we would test it, what productivity and fault tracking software we would use, what test servers would be utilized, and those sorts of details. It was about 5 pages, and had the correct cover page, and header and footer info, and

the things that auditors look for, and I tried to use bullet points and a diagram or two to cut down on the wall of text effect. It looked as aesthetically pleasing as a document as dry as a test plan could. I printed a copy, and hand delivered it to the lead programmer.

I walked over to his desk, held out the test plan with one hand, and put my foot against his desk drawer with the other. He gave me a funny look, then laughed and tossed it on his desk. After a few hours, he told me he had read it, and it was probably the least lame test plan he'd ever seen. A few days later, it was still on his desk, it hadn't gone into the drawer where project plans go to die! I considered that to be a great victory.

From that point on, whenever I have to create a test plan, I follow what I learned from James Bach. Actually, I don't just follow it, I take what I learned from experts, expand on it, adjust it to taste, and make it my own. I encourage you to do just the same.

Tracy's Thoughts

Tracy: You have to pay attention to more things when testing mobile apps since the environment, devices, wireless conditions and the users need to be considered. Having a more flexible outlook on mobile testing can save you lots of time, and you can create tests that fit the technology better. I've seen people far more experienced than me find far fewer bugs on mobile app projects because they spent so much time on traditional documentation and not enough time testing things that are important for mobile apps.

My secret? I planned with the technology in mind and tried to focus my testing efforts on actually testing, rather than spending too much time documenting. Instead, I created some simpler testing guides, and I document as I go with testing sessions. In my experience so far, this approach can be much more effective than traditional testing.

Another part of the secret of my success is to carefully research your users as well to get a better understanding of how your app will be used in the real world by real people. And once again, my favourite tools: using mnemonics prove to be quite efficient at generating ideas and being thorough in my thinking on every project. I start out with TAP IT UP and I SLICED UP FUN on mobile projects.

Concluding Thoughts

It is really easy to waste time on low-value activities on mobile projects, so please take even just a little bit of time to figure out how to approach your project. If I have an hour to test, I spend ten minutes figuring out a game plan. If I have a day to test, I spend an hour. If I have a couple of weeks, I spend a day. Whatever time I can spare makes a huge difference so I can focus on high priority test activities to help find the most important problems quickly. Don't put it off because you don't have time, make time.

Also, be flexible. Once you start working on your project, don't be afraid to change your plans when new information comes to light and your existing plan isn't working out as well as it could. Like anything else, you get better with practice, and there are a lot of factors on a project that are beyond our control.

Chapter 9: Test the Mobile Web

"What a tangled web we weave..." ~Shakespeare

Some of you may wonder why I have only devoted one chapter to testing mobile web apps, and why I have waited to talk about it in this book. "But the future is mobile web apps!" you say. Well, it very well may be, and we might see far fewer native apps at some point, but that day hasn't arrived, at least not yet.Furthermore, most of what I have written about native apps applies to web apps. While a web app may not utilize built-in sensors or other physical features on a mobile device, you still need to be aware of them when you test a web app. There is potential interference from different features on the device, whether it is from sensors, other programs, communications and user interaction that can have an impact on how your web app or web site performs on a mobile device.

For more on testing on the web in general, check out Hung Nguyen's book: *Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems.* While it was written before smartphones became popular, it has a great foundation for you to build on. I'll provide a bit of a technical foundation of how the web works in this chapter

to help your test thinking. Understanding a bit of history and how technology works will help you create more and better test ideas.

A Brief History of Web Apps

Way back in the early'90s, I was introduced to the World Wide Web (the Web or www). It was a new system that had been developed to link documents across multiple machines, using the internet and other technologies. Tim Berners Lee led a team that created a system that allowed for free and simple document sharing. The internet had been already around for over 20 years, and there were other collaboration models such as internet forums and Usenet. What was fascinating about the web was the availability of information from many different sources, on many different topics, hosted on computers all over the world. It was there for free!

It is hard to describe how amazing and revolutionary this was. I was a young student with a university computer account, using a text-based browser called Lynx. I suddenly had almost instant access to all kinds of useful information for research purposes. In the past, I would work through the library systems, find a journal article or book that seemed to be interesting, then go through a lengthy application and inter-library loans process where physical copies were mailed to me to utilize. Many times the information wasn't that useful, or it arrived too late to help me with assign-

ments. With the web, I could find out a lot more much more quickly.

I used the web primarily for university research, until a colleague showed me an online store. Back in the fall of 1994, I eagerly entered in my credit card information into a Lynx web browser and hoped that a rare and difficult to find CD (compact disc) would arrive to my home in the mail. A few days later, the envelope arrived, and I still own the CD, a version of Rachmaninoff Vespers sung by the Robert Shaw Festival Singers. It all seems a bit quaint now, but that was revolutionary, and it opened my eyes to the potential of the web.

I was soon introduced to the Mosaic and then Netscape document-based browsers, and I quickly learned about "hyperlinks", the usually blue, underlined words in a document that link to documents on one particular machine, or to others, that could be hosted anywhere in the world. Hyperlinking brought about a seamless front, and helped the web expand across the world.

In the early days of the web, we just had pages of a certain format that were uploaded to a server. Anyone who had the address of the server could go and view those documents on their own machines (called clients). We had a lot of issues with performance because by then we were using modems to connect to the internet, which allowed computers to communicate using telephone technology.

To connect to the web involved starting a modem, taking up a phone line (it couldn't be used for voice communication

while the computer was using it) and dialling a number to your internet service provider. It would take several seconds to connect, and any page you went to on the internet could take a while to fully load through your web browser.

With modern web browsers, we expanded beyond simple text documents, and added in more style and polish. We created documents using a markup language, called "HyperText Markup Language" or HTML which is a formatting language. It has its own reserved words and symbols that are transferred through cyberspace to computers, then translated and displayed by web browser programs on client computers. For security reasons and to keep pages small and simple, they were static documents with links to other pages in them. We added pictures, and eventually embedded sound and even video files, but any large files could take forever to transfer, download and render on a client computer.

As internet communications technology became faster and faster, the performance of the web improved. With that came add-on programs to make the pages more interesting and dynamic. One of the first and still most popular ways of making a web page more interesting than a plain document was the inclusion of JavaScript support. JavaScript was revolutionary because it allowed traditional programming power to be part of what were static web-pages. Now we could do processing in web browsers on client machines instead of doing it all on a server. This brought about a

much more dynamic web, and helped improve not only the user experience, but the performance of our web sites.

Web browsers started adding in more support beyond JavaScript so that programmers to write little bits of code that could be embedded in HTML web pages. so the page could do more than just sit there. There were also plug-in applications such as Java Applets, Adobe (Shockwave) Flash, and Apple's Quicktime. With plugins, you created a little program that was displayed in your web browser that could have complex animation and computing. The days of a static document that sat there and did nothing were over. However, now we had more complexity, and larger file sizes to download and be rendered on people's PCs. That provided more opportunity for problems and performance issues.

We got more sophisticated with web pages becoming more and more like the native apps we were used to using on our PCs by combining the bare bones HTML that made up a page, with Cascading Style Sheets (CSS) to help make them look more interesting, and JavaScript to make them more interactive. As technology has improved, today we have lightning fast internet connections that allow us to browse the web with almost instant responses, and we have rich media embedded in web applications. It is so slick and widespread, many people probably don't even realize that web pages and most actions you undertake with it require a round trip from your computer over networks to another computer. This adds a performance hit, and makes your

computer work harder to process and display things within your web browser program than many native PC apps that do everything on the PC without going out on a network.

There is a lot to like about storing web sites or apps this way. You can distribute things to anyone with a computer that has a special program for accessing the web, downloading web pages, and displaying them on your machine: a web browser. You don't have to get a lot of different people to download and install an app on their own machines, and deal with a lot of compatibility issues that crop up when you try to support a lot of different operating systems and hardware types. (Is this starting to sound a lot like our current mobile environment to you?)

In the mid-late '90s I started doing work as a web developer. I started with "static" web sites that were created by using HTML and CSS with images, and possibly a tiny bit of JavaScript in a series of web pages that I created and uploaded to web application servers. People went to the server address through the URL (Uniform Resource Locator) or web server/site address. They would then download these pages to their own machine, and the machine would display them exactly as I had saved and uploaded them. Their web browser made copies of those documents in its cache to help boost performance when navigating. This isn't very efficient though if you want something more complex, or want to perform business transactions, so we started using technology that would we could run on servers that would dynamically create the HTML files for users. Each user

could have a different variation, because each file would be generated just for them. That meant the content could be personalized and unique.

To create a web app, you need a development framework to run on an application server that is connected to the web. It has a programming language (often two or three), files to govern formatting (HTML, CSS) and usually stores information in a database. We used languages like JavaScript that were downloaded and run on the user's machine to make a web page more dynamic, then we created "templating engines" which were programs that created the web pages themselves, out of languages like Perl, Python or Java. There would be access to a database that stored important information. Several languages might come in to play to perform calculations and computation that was created in a programming language, like Perl, and then we used the common language of databases: SQL (Structured Query Language) to use the stored data. Now, this probably sounds like a lot of work, and it was, but it enabled many of us to create the foundation of the web as you know it today. It would be impossible to do most of what you probably find interesting today on the web without these kinds of apps.

Trouble was, these systems put much more strain on our servers. I worked at one company that created a job search engine, a real-estate service, and an amazing e-business application. We started out with simple servers built out of moderately priced workstations in our office, but as

traffic increased, our machines would slow down. At the time, it was common to buy one very powerful machine to run your web app off of. However, they were expensive. I remember just one server costing us over $80 000 for the hardware, and tens of thousands more for operating system and required software licences. A new, revolutionary idea came out where people took old hardware, and installed free, open source operating systems on them, and wired the machines together. These became known as "clusters". I, and colleagues of mine, started looking for old discarded computers that businesses were getting rid of. We found that with a lightweight operating system, and dedicating certain machines to certain tasks, and getting them to all work together, we could outperform some of the incredibly expensive single machines. This model has become a standard - it's easy to improve performance of web apps when you just buy another piece of hardware and add it to your management system.

Ok, that's enough ancient history.

Nowadays, we have mobile web browsing on smartphones and tablets which is extending to other sensor-based computing platforms like wearables and others. We have had this for years on feature phones, but it was often slow and cumbersome. With faster cellular and wifi networks and more powerful devices, many of us browse the web and use web apps on our mobile devices as much or even more than we do on PCs. Technically, we can go to any web site with a mobile device that we can with a PC, but formatting

might be wrong, or they may use plugin programs that your device doesn't support. Since the screens are smaller, and mobile devices have a singular focus, it can be hard to use some web sites. As a result, many web apps or web sites use special technology to determine what kind of device you are accessing them with, and then redirect you to a different, custom made version that is optimized for mobile devices. (These are often referred to as "m-dot" web sites, which are stripped down copies of an existing web experience. You will see something like an "m" for mobile, "mobile" or "touch" in these URLs.)

Other web site and app creators decide to take a different tack. It's a pain to have different versions of a web site or app to maintain, so they optimize for mobile devices. It will work well on PCs, and the smaller screens and different usage and interaction on a mobile device. There are still challenges though - cellular networks often have performance that is closer to modems on the early days of the web than the lightning fast connections we have grown accustomed to. The devices are less powerful than their PC counterparts and can take more time and processing resources to deal with media and other larger files.

Note: Don't be fooled by marketing numbers regarding clock speed, multi-cores, and memory. The architecture on these devices is very different, and has far more physical restrictions, so they do not perform the same as a PC with similar marketing specifications!

While mobile web apps have a huge positive upside, for

the same reason they did for PCs, there are a lot of issues to overcome. PCs utilize fast wired networks, they have a lot of processing power for their size, and there aren't a lot of size restrictions on files that can be sent to each user's machine to create their web experience. Mobile devices have a lot of different features and do not enjoy the same stability and processing power that their PC cousins do. There are a lot of performance challenges on the devices that tend to cause mobile web experiences to be slower than mobile app, or PC web experiences are. Furthermore, web technology for mobile devices is quite new, and with any new technology comes growing pains. (In fact, at the time of writing some large companies are discontinuing their web apps and replacing them with more performant native apps.) I am not seeing the death of native applications any time soon, but we are seeing growth of web apps for mobile.

These are all issues that will eventually be overcome, and some very clever app developers can already figure their way around many of them. The web remains an incredibly attractive platform for application distribution in the mobile space, so mobile testers should be prepared for testing. Furthermore, many native apps have web components embedded in them, so your testing will have to take it into account more often than you might have thought.

Web App Components

Web apps and web sites differ in a couple of ways. Applications tend to offer more features, and interaction. You can

log in, use your own profile to do things that are unique to you. Or you can purchase things, get involved with different information sharing or social interactions, or gaming. Web sites tend to be more static, and informational. As time has gone on, this has become more blurred. Web sites have embedded applications in them to help with advertising or better user experiences, and web apps can have more static content. For the purposes of this chapter, I will no longer make the distinction, I will just talk about web interaction through a mobile device as using a "web app".

It gets even more murky since web site information can be incorporated through a native application, and some native apps that you buy on an Application Store are actually web apps that use web technology. It can be difficult to tell whether an app is using native or web-based technology. You'll need to ask the designers and developers to find out for sure.

How They Work

Web apps require an internet connection to work. Let me rephrase that: any web app or site *completely depends* on good networking conditions to be able to function *at all*. This is an inherent weakness and a convenient strength, but it isn't taken into account very often.

Have you ever tried using an app with Airplane Mode turned on? If you start it and it fails because it has no connection to the internet, then you know it has some web components. Web apps require an internet connection

because all the smarts and content reside on a server somewhere else in the world. There is very little installed on your machine, just something to facilitate a network connection, and a program to display and allow you to interact with the app. The actual brains and guts of the app can reside anywhere else in the world. It might be stored on a computer down the hall, down the street, in a different city, or a different country.

Web Browser

Many web apps are only available through a special program on your device, a web browser. If the only way you can access the app is by opening a web browser, such as Safari, Chrome, Dolphin or Internet Explorer, then you are dealing with something that is 100% web technology.

A web browser is a powerful program that manages downloading and uploading of information from your device. It deals with network connections, translating information from different sources, processing what it receives and displaying it to you. Just about any time you touch something like a link, button, or image in a web browser app, it is likely sending information to the server, which processes what your inputs were, and sends something back.

Watch for a processing icon when you are using your web browser. That almost always means that it is communicating with a server somewhere else.

Web browsers don't just display HTML pages that are formatted by CSS docs, with embedded files (images, video, sound, etc) but they have special programs that help create a better experience.

On mobile devices, many web browsers reserve certain gestures and touch inputs for themselves. This can cause collisions or strange behavior if we are using those gestures in our app.

Special Programs

JavaScript engines are a huge part of web browsers, because web developers use it a tremendous amount to create great web experiences. They are complex programs that often have to handle security, computing, rendering and user interactions. They can also send their own requests to app servers. Every line of JavaScript code has to be sent to your device over the internet, downloaded by your machine, constituted into something coherent, then interpreted and executed. With complex activities that create even cooler user experiences, comes more code, and in turn, larger file sizes and downloads.

There are others that you can encounter, such as Adobe Flash apps, Apple QuickTime apps and MicroSoft SilverLight apps. Different platforms support different versions, so unlike on a PC where you just download a helper app to play a QuickTime movie when prompted, some devices will not support certain formats. This is something to watch for when testing mobile apps.

Settings, Cookies and Caches

Web browsers can personalize and optimize your user experience by allowing for some simple features that are saved on a device. Settings are the simplest, they allow you to customize your web browser itself; you can save usernames and passwords and it fills them in for you, and you can save your favorite web sites to speed up your browsing experience.

Cookies are an interesting tool for web apps. Web technology and web browsers severely limit the kinds of files that a web app can store on your machine. This is done for security reasons, because you don't want just any application to be installed on a client machine. This could be exploited by people with nefarious purposes. Therefore, to allow for a bit of user customization, web sites and apps can store a small text file on your machine, called a "cookie" which has some simple information about your preferences and user habits written in it. Once you have visited a web site or app, the cookie is stored, and subsequent visits will be slightly customized for you. The web browser does this by looking for a cookie file when you go to a different web address. It alters the web page that is loading on your machine based on that information. Web browsers allow you to delete these cookies, so your next visit is fresh.

Speaking of temporarily storing files, a web browser also has a feature called a "cache". This is temporary local storage on a client's machine that is used exclusively by the web browser program. When you visit a web site or use a

web app, all the information it downloaded is stored in your file system, or in a special database the web browser uses. The next time you go to that site, it reads the local copy that it has stored, which reduces that round trip traffic and speeds up performance. Once it has been overwritten, or deleted, you start fresh again.

Components You Can See

When you load a web page, it has text, images, media, and a lot of formatting. We have fancy fonts, pretty colors, and pleasing images to create a great experience. We can interact with a lot of objects on that web page:

- Text hyperlinks to other information or features
- Images for display or as links
- Buttons or arrows as links
- Media such as video or sound file players
- In games or entertainment:
  - Gesture-based interaction - drag and drop, move items, add remove items, etc.

What You Don't See

How we create that web page is often invisible to an end user. As I have mentioned earlier, a lot of technology combinations come into play to display that page to us on our devices. The parts we see are really just the tip of the iceberg. Most of what constitutes a web experience is hidden from the end user.

Networks

A network connection is what makes the web work. The web is built on the internet, which is a series of technologies that allow for many different computers to be connected together. Networks rely on special hardware and software on every machine that can connect to one another. In the old days, when you bought a PC, you had to go out and buy a "network card" of some sort so that you could connect it to other machines. Different network technologies speak different languages, and use different tools and processes, so you had to be careful to buy the right one. Nowadays, our devices come with networking built-in, but it is still important to understand some basic components. Here are some very simplified network concepts to understand:

- Hardware for network connections, communication, security
- Software to manage all of the above
- A protocol, which is a special communication language that all machines on that network need to be able to support and understand
- Specialized hardware in service centers to route and manage the traffic from multiple machines
- Specialized software and systems to manage unique machine addresses, information storage and optimization tools

On mobile devices, this picture is more complex due to cellular networks and different technologies that are used to

support the web running through them. They have a harder time with these dependencies due to slower connection speeds, changing networks while on the move, differing signal strengths, and sources of interference.

If you take your device, and open a web browser, and type this in the location bar: "http://www.google.com" and tap "Go!" what do you think is happening?

First off, that "http" stands for hypertext transmission protocol, which is the communication language of web networking. It is a subset of the TCP/IP family of protocols, Transmission Control Protocol/Internet Protocol. Each device using TCP/IP, and therefore each device on the web, has a unique address, called an IP address, or Internet Protocol address. This is often shortened to: "IP", or "what is the machine's IP?" The http protocol supports requests and responses to a server, and your client machine will have a special address, which is numeric. There are different formats for these addresses, because much like telephone numbers, you can run out after a while.

The rest of what you typed makes up a URL (uniform resource locator), which is an address. "www" stands for World Wide Web, which is the part of the internet we are using. "google.com" is an English version of a numerical address that points you to the Google system. This isn't just one machine, but many machines can be subsets of this address.

When you tap "Enter" or "Go" or whatever it is to make your web browser go to that address, your web browser

will reserve space on your networking hardware/software, tell it to make an outbound connection, and it will then use whatever networking technology it is currently utilizing to send out that address. When something on that network responds to your networking system, your web browser will initiate a connection, and download the contents that are on that connection. All of this occurs over the network first, and is then downloaded, temporarily saved, rendered and displayed to you in the web browser.

If you are using a banking app, or entering in a credit card to make a purchase, you will notice that the "http" prefix is replaced with "https" which stands for a secure version of the protocol. It uses encryption, which is a way to obfuscate communication so it isn't easy to read by someone who might be elsewhere on the network and intercepting your information.

If you mistype a URL, or go to a page that no longer exists, your web browser may get no response, or an error code from a server. It will then try to interpret this and provide you with helpful information so you can fix it.

Does all of this sound complicated? That's because it is!

That simple action of going to a website through your web browser requires masses of technology all working properly. Various systems handle broken or slow connections, or incomplete communication and retrying, and use various ways to try to work around the problems.

The bottom line with networking: without a network connection, there is no web. At least for you. It is absolutely vital for any web experience to work, it must reliably connect and interact with other machines on a network.

Servers and the "Back End"

Servers and related technology are responsible for creating the content that is sent to your machine each time you interact with a web page and it needs to be changed and updated with something new.

As I mentioned above, web apps work by having either a set of files saved on a machine connected to a network that are made available to others via the world wide web, or a program that generates dynamic, unique versions of these files for each client machine. Servers have their own special addresses, and allow for limited access to their files and systems. You don't want everything on a server available to people, because they could accidentally delete or move something, or they could take advantage and install something harmful on the machine.

A server usually has special programs that create web experiences for other devices and people. If a web app is popular, you may have several machines with different jobs to help manage performance:

- App server to create web experiences for client devices

- Database to store program and user information
- Performance enhancement , such as caching machines
- Networking switches and balancers to quarterback how and when different machines are used

A common web application development framework style is called a "model, view, controller" (MVC) which is a programming style that divvies up the code of an application into sections. One handles the creation of the things a user will see on their device (the view), another manages user interaction (the controller), and another area has data that governs the way the application looks and feels and responds (the model.) Each of these areas can be made up of specialized programs, and these systems themselves are quite complex. One incredibly popular web framework that a lot of mobile apps use for their backend is called Ruby on Rails.

Other acronyms you may come across are LAMP and CRUD. CRUD is a term for interacting with a database and each word: Create, Read, Update, Delete stands for what an application can do with data that is persisted (stored) in a database system. Web frameworks use database backends a lot. In fact, one popular combination is LAMP: Linux server operating system, Apache web server, MySQL database, and PHP programming language. LAMP is incredibly popular because it utilizes free, open source tools. The Apache web server in particular has a long, influential history with the web.

Web apps developed on servers are quite complex things. They can span several technologies to create one web experience:

- Different server hardware for different purposes
- Databases and disk storage
- Network systems and optimization
- Different programming languages, and media

As I mentioned before, watch for a progress wheel. When you see it, your web app is interacting or attempting to interact with this complex system.

Common Technologies

When it comes to the app on your device, and how it is displayed there, you will likely come across these technologies.

HTML, CSS and JavaScript

HTML as I mentioned above, is what allows us to create documents that can be used over the web. At the most basic, the documents have a ".html" extension, and are easily viewed in a web browser. HTML is a markup language, which is a simple formatting system. In the old days, we used to use this a lot before we had powerful word processors that do it automatically for us. In fact, I am using a simple markup language as I write this book.

To use a markup language like HTML, you need a text editor to create the file, and a web browser to view it. If you want to try this for yourself, open up a text file, and enter in the following:

<HTML>

<HEAD>

<TITLE> Hello Mobile Tester!</TITLE>

</HEAD>

<BODY>

<P>

I am an extraordinary mobile tester!

</P>

</BODY>

</HTML>

Save it, and open it with your web browser. You won;t see all the special words that are in between the less than and greater than signs such as < >. (These are known as tags, and there are a lot of them.) Those are the markup, and the stuff you put between the markup words and symbols is what is displayed. Each of those symbols tell your web browser how to display the page to an end user. This is plain, it just uses default fonts and colors, but you can get very creative.

CSS is similar to HTML, but it is a formatting language. While you can have a web page with just plain HTML, CSS

depends on HTML to work. It is incredibly powerful and many of the amazing designs you currently enjoy on the web were created by talented CSS designers.

JavaScript, as I mentioned above, is a special programming language that is supported by web browsers. You can embed it in HTML pages, or just have references to JavaScript files in them. Your device has to download them, interpret them and display them to you. They add a different dimension, because they can support some level of interactivity that can take place on the web browser. JavaScript can do some animation and computation on a client device which saves the round trip to the server.

HTML5

HTML5 is a new version of HTML, and comes after several versions based around 1, 2, 3 and 4. HTML5 has new tags and technology features that support rich client experiences:

- Multimedia
- Animation
- PC app-like interaction:
  - Editing docs
  - Offline mode (can still work to some extent without an internet connection, but will need a connection to update to a server eventually)
- Location services

- More and better storage over the simple cookies

A lot of these should sound familiar to mobile technology you have read about already in the book. Rich client experiences are those that are polished, interactive and have aspects like embedded media types, a lot of support for user actions and inputs, and high resolution images and animation. They are richer than a pure HTML web experience because they look better and you can do a lot more with them.

There are a lot of sophisticated aspects to HTML5, including APIs (application programming interfaces) and special libraries to interact with. As a result of this increased power and inclusion of features we enjoy in native PC and mobile apps, it is rapidly becoming a platform of choice for mobile web apps.

Many mobile apps have this combination of technology:

- HTML/CSS for formatting (part of HTML5)
- HTML5-specific features for rich client experience
- JavaScript to enhance both formatting and rich client experiences

There are dozens and dozens of different web development frameworks that support this combination. Some are specialized, getting the job done on client devices with just HTML5, and others do it all in JavaScript, using it to generate web pages on the fly on a client machine.

Web Browser Libraries for Apps

Remember above when I said it can be a bit complicated knowing whether you have installed a native or web app that you downloaded from a store? Many people feel that if they download an app from an application store, and install it, it is native, not web. Conversely, if you can only access an application through a web browser, it must therefore be a web app. It turns out, only one of these statements is true.

To make things complicated for us testers, there are simplified, stripped down versions of web browsers that programmers can use in native app development frameworks. They can create a native app that is essentially a simple web browser that gets all of its information, content and computation from servers over the web. These are sometimes referred to as "shell apps" meaning that the native part of the app that allows it to be uploaded to a store and installed on your device is just a shell. The guts of the app is using web technology, stored on a server somewhere that you have to access in a built-in web browser.

In other cases, they may have a native app that has many features that work on the device itself, but uses a web browser library in certain areas of the app.

If you have to use a web browser to access the app, then you know it is web based. However, in many cases you won't be able to tell whether a seemingly native app is truly, or completely native or web-based unless you ask the designers and developers. As soon as you are testing a web-

based component in an application, that adds a different dimension and complexity to the situation.

Responsive Design

A relatively new technology for web apps and web sites is called "responsive design". This is an approach and toolset that allows for a single web experience to be served up on different device types. The web page you view on a responsive site will resize, but the content also morphs to a more singular focus as the screen size gets smaller. Images and other objects are also resized. It's a powerful concept and toolset, but it is still quite new, so it has some problems, particularly with resizing of objects:

Problems to watch with responsive design:

- Images may not resize properly on smaller screens
- Embedded media such as video players may not work, or may not resize properly
- Tables may not resize properly on different screen sizes
- Touch interaction may not be accurate, often due to poor workflow and interaction design
- 3rd party plugins such as ad, or content hosted elsewhere may not resize, or may not follow a responsive design
- Less common screen sizes may not render correctly - they may appear to be too large or too small for the screen

- Web browser updates breaking responsive design framework code

Responsive design is actually a new spin on something we called DHTML (Dynamic HTML) years ago. It depends on special code in CSS and JavaScript that provide alternative options for different screen sizes. Once your web app or site has determined the screen size, it will call certain formatting sections in CSS that are optimized for that size of screen.

Adaptive Design works similarly, but it uses code to determine the *device type* and then resizes based on that information. A tricky part of this is it is getting harder and harder to determine what kind of device people are using.

Both responsive and adaptive design are new technologies that aren't completely supported by web browsers yet. Many responsive design framework developers use loopholes or quirks in web browsers to mimic feature support that isn't there. When a new version of a web browser comes out, those loopholes or quirks may no longer exist, breaking functionality. We'll get there eventually, but in the meantime we need to test on different device types with different screen sizes, and we need to retest when there are updates to popular web browsers.

Mobile Considerations for Testing

With native apps, you don't have physical limitations and delays to worry about. It is all just there, on that device, so it is faster and more reliable. With a web application, there is a lot more complication. All communication to support a web app has to go through various systems and a communication channel between your device and at least one server has to be open, reliable and reasonably fast.

If you think about it for a moment, it will make more sense. Electrons moving around inside a device don't have much physical distance to follow. Communication with another device that is far away requires far more infrastructure, larger messages between systems, as well as distance, and the time for messages to travel back and forth takes time.

Mobile web apps hold a lot of promise, but what at first may seem like a superior choice over native apps comes with a lot of challenges. As I mentioned earlier, much has been written about testing web technology, so I encourage you to research that on your own. I will now focus on mobile-specific issues to be aware of when you are testing mobile web sites and web apps.

Mobile Web Challenges

While it might seem to be easier to develop web apps than native apps, there are plenty of challenges and potential problems that we need to test for.

- Smaller screens with different resolutions
- Web designers using "fixed width" elements in pages, so they don't resize for mobile devices
- Different hardware, lots of different web browsers to choose from
- Mobile devices are not as performant as their PC cousins - processing, rendering, and network interaction are all slower due to different architecture constraints
- Dealing with non-PC features: sensors, touch screens, different network types, and optimization modes to deal with low batteries and heat dissipation
- Understanding and incorporating mobile device features: cameras, location services, etc.
- Network latency, transitioning between network types, and handling problems
- Assumptions about unlimited bandwidth can backfire - poor performance, high costs if data usage goes over contract limits in some countries
- Far less plugin support like Flash, Silverlight, QuickTime, etc.
- "Always on", "always connected" expectations and usage of mobile devices

Consideration: Layout and Design

Mobile devices have smaller screens, which leaves less screen real estate for web sites and apps. Mobile designers and marketers say "simpler is better" for mobile. However,

when you are used to developing web experiences for PCs which can have large screens, it can be difficult to transition. "Fixed width" in HTML or CSS code can be used to optimize views for PC monitors, but can be a nightmare on mobile devices. You may end up with a web page that is far too wide for your device.

Mobile apps require a singular focus, because they need to utilize space on a smaller screen, and they need to be usable. Traditional web experiences are often multi-focus, which can lead to a complex design. At best, overly complex web designs can be confusing. At worst, they can be downright unusable. Merely shrinking an existing design to fit a mobile device screen size is probably not going to turn out the way we might hope.

Consideration: Touchscreen and Motion Interaction

Traditional web experiences depend on using keyboards for typing, and a mouse for pointing, navigation and certain kinds of inputs. A touchscreen is a very different experience, and it can be difficult to transition. A finger navigating and tapping is not always the same as a mouse navigating and clicking. Fingers have a larger pointer surface, and touch targets may not work correctly on images, links and other objects you have on a web app for use interaction. Furthermore, you may not support common input gestures that people are used to utilizing with other apps. Some web apps utilize optional inputs such as keyboard hotkeys, or

right click menus brought up by a mouse. These are not directly supported by mobile devices.

Now we have technology that doesn't require touch, we just have to move near the device. This has enormous implications for web apps because they may be overlooked during development and design. If the device you are testing supports motion gestures as well as the touchscreen, make sure to try them all.

Consideration: Different Network Connectivity

At the time of writing this book, one of the most difficult factors for web apps is the differing technology with wireless connectivity. Since a web app requires a network connection to function at all, we assume that the device accessing our web app or web site has an internet connection. However, unlike our PC cousins, wireless network connectivity is different on mobile devices. Wifi signals can vary as we move around a single hotspot. As we move longer distances, we move through and use various wireless technologies. Each technology has its strengths an weaknesses. Furthermore transition points between network types as well as in and out of wireless access dead spots are difficult for web apps to deal with.

Consideration: Accounting for Combined Activities

Traditional PC use has far less mobility involved than with sensor-based, mobile devices. We have been traditionally limited by wires: power cables and network connections. Laptops have batteries and wifi support, but they tend to be larger, and not used on the move like mobile devices. For example, I rarely see someone walking to the train station near my home while typing on a laptop, but I see people texting, gaming, or web browsing during their morning commute with mobile devices constantly.

As a result, our web designs tend to assume that people will be primarily focused on the computer. We don't think about different light, weather, network strengths, emotions, user motivations, people being in a rush, or using something at their leisure, or other technologies working on a device (think sensors and location services) at the same time as they are surfing the web. As I mentioned in an earlier chapter, we tend to integrate mobile devices with other devices, and our web usage patterns can be quite different than on PCs as a result.

Consideration: Coping With Platform Fragmentation

While it is tempting to think by that creating a web app instead of native apps we are eliminating our platform fragmentation problems, we are only trading one set of

problems for another. That said, a mobile web experience is a no-brainer, because it is much easier to get support on multiple platforms using one code base. With native apps, you have to develop for each operating system you support. That said, you still have to look at testing on different platform combinations, and you will need to work on your designs to ensure they work on popular platforms.

Mobile web platforms can include:

- Hardware model (smartphone or tablet)
- Operating System version and sub-version
- Web browser brand and model
- Network type
  - Wifi
  - Cellular networks (EDGE, 2G, 3G, 4G LTE, etc.)
- Secure or non-secure communication (http vs. https)
- Cellular provider

There are several web browsers available for each operating system. Each of them can behave very differently, especially if your web app is complex and uses a lot of different technology. Some are fast with JavaScript processing, others are great when dealing with large media files. You can also add different network strengths for each of the network types, as well as transitions in and out of each of them. Don't be surprised if you end up with a dozen different mobile browsers to test your app with.

Common Problems

I use the web on mobile devices a tremendous amount. I surf, I use web apps, and I search for things as I research purchases or services. Since I use it a lot, I get frustrated a lot. Here are some things that really get on my nerves, and I always watch for them when I test mobile web apps.

Ten Things I Hate About the Mobile Web

1. Poor design - I can't figure out what is going on, or how to find what I need
2. Poor performance - it takes forever for pages to load, or to respond to my actions
3. Poor input support - I can't figure out what to touch to interact with the app
    - Touch targets are poorly supported. Touch targets are either too small, or are slightly off, so I have to search around with my fingers to interact. They may also be too close together and numerous, causing me to accidentally enter in multiple inputs, or the wrong inputs
4. Poor stability. Many web apps crash a lot, and some of them even cause a reboot of my smartphone or tablet when they fail.
5. Too much typing and scrolling required. There may be long forms for me to fill in, and screens may all be much larger than my device resolution, causing me to have to constantly scroll.

6. Content is missing in the mobile version. If I go to the regular web site, I get it all. The mobile version is stripped down and missing key content.
7. Obnoxious nagging popup messages that obscures content and inputs, instead of letting me dismiss it and continue to use the web site without installing their app or clicking on an ad
8. Embedded content that isn't supported by my device, such as media that depends on certain players
9. The web app is a resource hog and slows down my device because it is constantly chatting with the server, downloading large files, changing content on the screen and calculating things.
10. Assuming I have a back button. If my device doesn't have one, I may get lost, or have no way to return to the home page if I am several pages deep.
11. *BONUS*: Not handling different network strengths, and transitioning from one wireless network to another, or dealing with network error conditions.
12. *SECOND BONUS*: Image carousels! What a waste of screen real estate, download bandwidth and speed, and processing performance!

Help! This Mobile Web Page is Too Busy!

I'll discuss some of these issues in further detail for the remainder of this chapter.

Usability

When I'm using a mobile device to surf the web or use a web app, I'm not doing it in ideal conditions. On a PC,

I have more patience because I have more space to work with, and a more powerful device. Mobility by definition means I will be moving around, so I will be in different locations, holding a device while standing, moving, or sitting, in different lights, with different weather, different background noise and distractions, and on network connections of various speeds and reliability.

I'm not using a mobile device the way I use a PC. I'm not sitting down gazing into a large monitor, in my comfy-womfy chair listening to Mozart, sipping bourbon and stroking a puppy. I'm checking a price in a crowded supermarket, trying to find a new flight in an airport waiting area, or trying to catch up on news while standing in line outside in the cold weather, or waiting nervously in a doctor's office. As a result, web usability on mobile devices is incredibly important, just as it is with native apps.

Unfortunately, due to the web's PC legacy, much of what we try to interact with on the web was not designed with mobile in mind at all.

Poor Mobile Designs

Usability problems top the list of common problems I encounter when using web apps on mobile devices. Part of it has to do with the way web pages have evolved over the past twenty years. In the beginning, they were very plain, often with mostly text, the odd hyperlink and picture, with white backgrounds, or some drab, inoffensive color. They then became quite garish with overuse of images, especially

animated images that would blink and move around. Once business and other professional organizations moved to the web en-masse, the professional designers helped create the web experiences we know today. However, we are cramming more and more into web pages.

Ten years ago, the main body of the page took up more real estate. Now, that has shrunk, and we have banners and ads on the top of the page, with navigation elements. On the left side, we often have more navigation elements, and we cram in ads and other information on the side. At the bottom of the page, we have more navigation, for going from one page to another, or for even more pages or features in an app. We also tend to have larger screens with higher resolution on PCs, so we have had more room to play with. Mobile devices shrink this space way down. A complex web page that works on a large PC monitor will be a nightmare to deal with on a smaller mobile device. Any distractions away from the core purpose of a web site or web app will be frustrating on a mobile device.

When we use the mobile web, we don't sit and surf the same way we do on a PC. That's mostly because we may not be sitting, we may be mobile. If we are sitting, we aren't necessarily in a comfortable position, so we often focus on specific tasks: research a new purchase, find out information to solve a harmless argument when out with friends, or to look up a fact or name we have forgotten. We want to find things when we're out and about. Does the store we are looking for have a clear address posted? Does

it sync with mapping software on my device? Mobile users need to go to your web site, use it for its core purpose, and move on. If they have to spend too much time to utilize core features, or they find it difficult to use, they will get frustrated.

If mobile users are spending longer periods of time on your website, or web app, the content, features, entertainment and functions need to be usable for that usage as well. For example, many people post and interact with others on social networking sites, and online forums. Others provide access to TV shows, movies, or gaming experiences.

When web designers jam in a lot of information into a web page, it can be unclear on you can interact with. Navigation controls may be in a spot that is difficult to see on a mobile device. In many cases, navigation objects may not load on the screen and you may have to scroll to find them. Some web sites have navigation controls on the bottom, and a mobile user may just navigate away before finding them.

Sometimes a web app or web site might be optimized for a mobile experience, but other elements may not be. There might be advertising, or images and video hosted by other services that load slowly, or popup over top of your web site, obscuring it, and preventing you from interacting with it. Irritating popup ads that aren't easily dismissed can make a web site unusable.

Sometimes web developers use device detection create a pop over ad to encourage you to download their native mobile app. If you can't close it, it might that obscure view

and prevent inputs. I really don't want to install their app now!

At the time of writing this chapter, there are many, many examples of pure usability on mobile devices. For a lot of current examples, check out Brad Frost's WTF Mobile Web[32] for screen captures of web designs gone wrong.

Poor Performance

Usability problems are annoying, but poor performance can be truly rage inducing. I can't even delete the app, all I can do is look at my frozen device in helpless anger. Sometimes the web browser app will crash, providing immediate reprieve, but in the worst cases, poor performances can cause my device to restart.

A standard for web page load times is around 2 seconds. I'm not sure where it originated from, but I've been hearing it for years. In the late '90s when I started working on web applications, we were obsessed with optimizing page sizes so that they would download and render faster on people's machines. We had to assume that people had smaller, lower-resolution monitors, and were using slow modems on dial-up. We compressed images, optimized JavaScript, and reduced our features so that everyone would have the best web experience they could. A few kilobytes counted.

In fact, we got so obsessed with it, we even converted our files to have all the code and content on one line, just

[32] http://wtfmobileweb.com/

to see if it would speed things up a bit. Nowadays, we don't think about it so much anymore. We assume people have fast machines, with large screens and great network performance. On mobile devices this is not the case.

Keynote Mobile have done some research in the page load time area for mobile devices, and have found that consumers tend to expect sub 3 second page load times. Need for Speed: Smartphone and Tablet Users Hate Waiting Even Three Seconds to Load Websites[33]

Why do web pages and apps suffer from poor performance? It turns out that they can be quite complex, and require a lot of information to pass between a server and a client web browser to operate. Here are some common reasons why performance can be poor:

- There are too many server requests (the app is too chatty)
- File sizes are too large, or there are numerous files to download
- There is too much processing and rendering once the page has been downloaded and a user interacts with it

Let's explore each of these in a bit more detail.

Too many requests:

[33] http://www.techvibes.com/blog/need-for-speed-smartphone-and-tablet-users-hate-waiting-even-three-seconds-to-load-websites-2012-08-07

Imagine that every time the page on your screen needs to update, and display another page, or new version of that page, a round trip has to occur over the network. A message is sent from your device to a server, over different networking technology, then that request is processed by a program on the server, a new message is formed and sent back to your device. Now your device has to take this message, process it, and your web browser program displays the new version to you. Does that sound like a lot of work? Well, it is.

With modern technology such as HTML5, these requests can be sent off without user intervention. There may be a timer in the code to update portions of what is on your screen, or it may get new information from news or other information services. The app may work with social networking apps and services as well, which may require new information each time something in your social network is updated. These are very handy features, but they can really stress a mobile device to the limits. It is difficult for the device to handle a lot of web requests, and to render the new version of the web page on your screen. If too much happens at once, the entire device may slow down considerably.

File size problems:

At its simplest, a web app or web site is one file in a format that is recognized by a web browser program. However, that file can have a lot of other files associated with it that also need to download for it to work properly. That

web page file that is the first thing you see when you are using a web app will have image files, possibly media files such as video, and code that needs to be run on the client side, such as JavaScript. So when you load that web app, there is more than one file getting downloaded, interpreted, rendered and displayed. If some of these files are very large, it can take forever for a page to load because of objects and items that it requires haven't downloaded yet.

Larger image and media files mean that we have a better experience with a web page or app. Images and video are clearer and have more definition when they are larger because they haven't been compressed as much. Code files that are larger have more features, so we can do more with the web app or site. There is a trade-off though, and large files on cellular or wifi networks can slow down performance.

Here are some common file size problems:

- Physical size in bytes: the file is so large that it slows down network activity and browser rendering
  * Media files such as images or videos are too large
    - 3rd party ads are too large
    - Large JavaScript files:
      * Helper libraries often have mobile versions that are smaller. Coders may have forgotten and used the regular libraries.
      * There may be a lot of code that isn't used, but it's convenient to just send the whole thing.

The size of web pages coincide with the speed of technology and increased bandwidth. Modern PCs and supporting networks handle larger file sizes. However, with mobile devices, we are often back to the early days of the web. Slower download speeds, less bandwidth and more errors.

Another problem is that we just have too many files being sent with each web page request. This is easy to do when we use development frameworks, and include all the libraries that come by default. Now imagine compounding the size issue by having numerous files being downloaded and rendered.

Too much processing and rendering:

Modern web pages and apps do a lot of work within a web browser on client machines and devices. This can save round trips to a server, which can improve performance. It can also create more engaging experiences by using things like automation, or visible feedback to human interaction within the web page or app. However, remember that a web app or web page is accessed through a program, a web browser.

That web browser is using resources on your device to operate, and the page or app within it that you are accessing also requires those same resources. There is another trade-off here, we can do more on the device with special code and programs that our web browser can use to help create a richer web experience, but with that comes more resource requirements.

One simple processing and rendering hog is the currently popular carousel feature. This is an animated set of pictures and text that scrolls across the top of a web page. On many web sites, a mobile device will be trying to download all the images, and the code to make this work, and once it has finally rendered the first image, it may already be on the second "slide" in the carousel, needing to download and process some more. In the meantime, I as a user am merely trying to look up important information, but I can't seem to scroll past because the carousel is hogging resources.

I've also had web pages or apps freeze up completely when too many events were triggered, and the app was doing a lot at the time. My web browser is chugging along, and my device is managing network connections, battery optimization, location services and sensors, while a web page is kicking off a lot of code and trying to do something fancy. Instead of a rich client experience, I have frequent freeze ups and a slow, frustrating experience. Rendering new images or creating animation can be expensive, so when it is done on a web page, or within a web app, it needs to be done as optimally as possible. If we as designers and developers don't take other processing demands that could be going on at the same time on the device, we will likely overload resources.

The differences in performance between desktop PCs and mobile devices can be shocking. I read a recent report by Spaceport who had some interesting findings:

> "The fact of the matter is that even the newest,

> fastest, most modern smartphones in the world, running the latest operating systems and using the most up-to-date browsers, have HTML5 performance that is *many times* slower than on modern laptops and desktop computers. In some cases, it is *thousands* of times slower, and other times, high- performance HTML5 graphics techniques are simply not supported at all."
>
> "...We found that when comparing top-of-the-line, modern smartphones with modern laptop computers, mobile browsers were, on average, 889 times slower across the various rendering techniques tested. At best, the best iOS phone was roughly 6 times slower, and the best Android phone 10 times slower. At worst, these devices were thousands of times slower. Developers should heed these findings in the near term. Nevertheless, with HTML5 technology improving along each iteration, we are confident that highlighting the current deficiencies can only help HTML5 technology get to where it needs to be."

From: Spaceport PerfMarks Report II - HTML5 Performance on Desktop vs. Smartphones, May 2012[34]

[34] http://spaceport.io/spaceport_perfmarks_2_report_2012_5.pdf

Poor Handling of Network Conditions

At the time of writing, one of the biggest problems I encounter on mobile web sites or apps are due to designers and developers not handling different wireless network conditions well. It seems that many web developers assume my mobile network connection has massive bandwidth, very low latency, no connection outages or errors, and is consistent over time. Whenever many apps or web pages encounter a problem, they don't handle it very well. They crash, they freeze up, and in the worst cases, the server on the other end actually goes into an error state as well.

Web pages and apps don't seem to be able to handle differing network connection strengths and speeds, and tend to work well in ideal conditions. When we're mobile, we move between networks, in and out of dead spots where we have no network access, and we get different speeds and latency. Latency at its simplest just means that we might get longer delays between messages. In some cases, parts of messages may not even get through if the network is slow and doesn't have a strong signal at your current location.

To test how a web app or site handles network issues, I find a form or something to submit to the server, and I test the following conditions when I submit a request to the server. What happens when we:

- Have poor wifi signal strength
- Transition from wifi to cellular network
- Transition into a dead spot

- Transition out of a dead spot
- Transition from a cellular network to wifi
- Move from one wifi source to another

In one case, a file submission from a web app during a transition from wifi to a 3G cellular network, done on the move, caused the app to end up in an infinite loop error condition. We also found out that it did the same thing to the web server, preventing any other people to access it. It was a very serious problem caused by two programs that were not designed for this type of network condition. Both tried to go into error modes and recover, but got stuck. It was very messy to clean up, because even if restarted the device, as soon as we went to that web address, it would go back into an endless error mode.

Too Much Scrolling and Gesturing Required

When I try to use a web app that is poorly laid out on mobile devices, I do my best to navigate. However, I may completely miss information if the screen is wide horizontally. That's because I may need to use a horizontal scroll rather than a vertical. If I have to use both horizontal and vertical scrolling to find what I want, that can be very difficult. Too much scrolling in one direction is bad enough, but having to combine directions is difficult. It's easy to miss what you are looking for.

Some web sites are optimized for mobile devices, but lack a singular focus, or use a layout that is too large for my screen. Even though they support mobile views and gestures, I may have to gesture too much to get what I need. If I gesture over and over, I might accidentally go past the feature or information I need. Core, important information and features that mobile users will use the most need to load first and need to be easily accessible from the front page.

If I spend more time on gestures, I will have less brain power and patience left to actually use your app. If I have to gesture and scroll too much, I'll probably lose patience and move on.

Input Problems

A web site designed for PCs may not be easy to interact with on a mobile device. Web controls such as clickable hyperlinks, images, and the visible feedback that provides hints to users as to what they are doing was first designed for keyboards and mice. Just because it works well to navigate a pointer and click on objects with a mouse does not mean it will translate well to finger interaction on a device that depends on gestures. In fact, many web sites will need to be adjusted for touch gesture interaction.

Touch targets may be different than the area that can be activated or clicked on by a mouse. In some cases, they may be too small, or may not be on the object itself, causing the user to use trial and error to figure out just how far away

they need to be to interact with a control on the web site. I've tested web apps that required me to go up and to the right and slightly off of the menu bar so I could interact with them. I even had to move slightly off of hyperlinks, and some areas had touch targets that were so tiny it took me forever to figure out that I could even tap them to activate something in the app. This was confusing and frustrating.

In other cases, there are too many items or objects I can interact with that are too close together. That means that if I try to interact with one of them, I may interact with the wrong one unintentionally. This is sometimes called the "fat finger" problem. A user tries to gesture or tap on one item, only to interact with multiples, or press on the wrong one.

One of my favorite news web sites updated their web app a couple of years ago. It was clearly designed for tablets, because on my smartphone, I would constantly fat finger when I tried to open up a news story. Other items were placed very closely to the news story headline, which I would tap to load the story. Instead, I would either accidentally tap on an ad, which would take me to another web site, or I would accidentally load the story video, or image. It was frustrating to try to just load the text of the story, which was all I wanted to do. I was on the move, so I wanted to read the content and save my cellular data usage for something else. It could take forever to turn off media and go back and try to load the story, only to tap

on something else, which would load something else that I didn't want to see. Thankfully, they fixed this problem soon after the release.

Another issue is a lack of visible feedback when a user uses gestures and inputs on a web site. Many of the animation that provides users with feedback is related to mouse events: mouse over, mouse click, mouse off, etc. Web sites often have animation, or some other visible cue to show users that they can actually do something, or where they are on the page with their pointer. This special code may not react to touch inputs, leaving users feeling lost. "Hmmm… I tapped that thing, did it do anything?"

Too Much Gesturing Required

To really wear down mobile web users, make them gesture and type a lot on a touch screen. They will get frustrated and may just stop using the app. I have an example from this morning. I was sitting in a lounge chair, drinking coffee and reading email on my front deck, enjoying a sunny morning. I was catching up on my email on my smartphone, and a colleague asked me to go and vote for an online contest. I wanted to help them out, so I clicked the link, and went to go and vote for their project.

I was confronted with a long, long online form. There were dozens of questions with a choice to click in each area, and a form to fill out at the end with information. Lots of tapping, followed by lots of typing. What do you think

I did? If you guessed: shut down the browser and didn't vote" then you win the prize.

It is incredibly irritating to be forced to fill in a lot of information when using a mobile web site or app. The first place that can drive your users nuts is to have a really long URL, or web address. Or if it isn't long, make it fiddly or hard to read. ("Are those "ones" or the letter l?") If I have to type too much, or keep retrying until I get the actual web address right, I probably won't even make it to your web app. Long, weird, or fiddly URLs are a non-starter.

Long forms with a ton of input fields are a standard for signing up with a new account for a web app or e-commerce system, or for making a purchase. I have to enter in my name, salutation, address, credit card information, and so on. This can be overwhelming, and much of this information could be auto-populated, or even removed. As Mobile First[35] author Luke Wrobelewski says: "When it comes to mobile forms, be brutally efficient and trim, trim, trim."

Forcing mobile users to gesture and type and gesture and type and gesture and type some more is painful. I have a native app from one company installed on my devices, and I use a web app from a competitor. The native app requires 3 gestures to get the information I need. The web app/web site requires over 50! I have to tap over 20 times just to type in their URL! I always count gestures when I'm testing for the web because they can get out of hand.

[35] http://www.lukew.com/resources/mobile_first.asp

Content Problems

In some cases, designers and developers create two different versions of their web app, and use code to detect your device. They may have a mobile version that they redirect you to. Sometimes, important information is missing from the mobile version that you can get on the PC version. There's nothing worse than knowing you have a good resource, going to the web site on your mobile device, and not being able to find what you want in the mobile version!

Localization Issues

Spacing and display issues are often problems, since most web browsers can support different languages. A word, or several words translated may end up different sizes. For example, one word in English might be much longer in German, and three words in Finnish might translate into five words in English. Also, some letters in different languages can be different widths. Some of them even join letters together, like a and e, and o and e joined together. These are called "ligature characters." When the word or phrase is translated, it might not fit in the small space that is allocated for it. This is especially problematic with error messages. Another area to watch for special characters is in the URL. You might be able to type in a web address that uses a special character, only to have something break when the web browser or your web server try to resolve it.

I have also found some JavaScript libraries and other web development tools not support special characters. One web

app used a JavaScript engine that would crash or simply remove any characters that weren't in English. Since Canada is a bilingual country, any of the several French language characters with accents and other symbols on them would cause problems with the web app.

Any area of the app that requires display, printing, computation or processing can cause it to fail in strange ways. User input may not show up, or it may be saved and displayed incorrectly. Print jobs may look different from what you see on a screen, and translations may be wrong, or inaccurate. Numbers, date and time ands measurements may have completely different formats from one language to another.

3rd Party Issues

Web pages and apps are complicated beasts. At any given time, a web app is essentially a web page with some code, media and other things plugged into it. Web apps, like web sites, have several main "pages" that can be accessed through a URL, but web apps are dynamic - programs create the content. To make them even more rich in use experience, or to help them be more profitable, more features are added in. In fact, some web pages are a pastiche of several pages from different sources glued together.

In some cases, an entire section of a web page is developed and hosted by someone else altogether. The danger here is that while our web page or app might be mobile-friendly, a 3rd Party we depend on may not be. Your version might

be just dandy, but someone else might be trying to do too much, or their images and media are far too large, and obscure your mobile-friendly content. Advertisers are a common source of this problem. However, on larger, popular web sites or apps, different sections of what you see on the home page may be owned by different departments, and may have different code and content served up from different sources.

What might appear to be simple and straight forward when you load up the app in a web browser might be a complex collaboration between departments, companies and technologies. When just one of these isn't mobile-friendly, it can hurt performance, the user experience, or cause the entire web experience to be compromised.

Other 3rd party dependencies are from within the technology stack that designers and developers use to create web experiences. Many programmers use special frameworks for creating web apps, and they utilize plugins and special libraries to be downloaded and utilized by the app when you go to that web address. Some of these are not very mobile friendly; they may be large in size, or use technology for rendering and display that doesn't work well on mobile devices. As I mentioned earlier, many of these frameworks have mobile-optimized versions, and a programmer may forget to include that for mobile experiences. Instead of a slimmed down mobile-friendly version, they slow down your experience by forcing you to download the version for PCs.

There are also dependencies on well-known rich client plugins such as Flash, Silverlight and Quicktime.Each of these technologies require you to download and install a player within your web browser. Many mobile browsers do not support this, or will only support one of the platforms. Many web experiences that are fantastic on a PC do not work at all on popular mobile devices, because those devices do not support this player and the technology. (This is one of the reasons HTML5 is becoming much more popular, it is an open technology that has almost universal support with web browsers.)

It can be painful for companies who have relied on plugin technology for a great client experience to try to replicate this on mobile devices.

Hogging Resources

As I've mentioned elsewhere, resources on a mobile device are more scarce than their larger PC cousins. It is quite easy to dominate a device with a web app or site that has a lot going on. Here are some things to watch for when you are using a feature-ful web app:

- Run down the battery quickly when rendering a lot of images or media, or animation
- Run down the battery when requiring a lot of network traffic between device and server when users are on cellular networks

- Too many page redirects causing the web browser to reload a lot, which takes more time and resources on a mobile device
- Hogging memory and processing power by doing a lot of client-side scripting, such as JavaScript
- Too many background activities on a web app can slow everything else down (Getting data feeds or updates behind the scenes)
- Poor media integration (camera, video, sound, etc.) can be difficult because some of these features require us to load a new app. We don't have players and plugins to watch all videos through a web browser, for example.

It can be irritating to try to use a web app and find that your device heats up, loses battery power and slows down, interrupting the usage of other features you enjoy on the device.

Memory leaks are a programming error that can be common in web apps. If an app starts to slow down after you have been using it for a while, it might be using memory and not freeing it up after it is finished with it. If you notice an app slowing down over time, let your programmer friend know right away.

Compatibility Issues

As I've mentioned elsewhere in the book, there is a massive amount of fragmentation in the mobile space. A web app is

one attractive way of dealing with a multitude of devices and programming languages. While you get compatibility on multiple devices by using their web browsers rather than multiple native apps, you still have to test them on popular web browsers. There are dozens of web browsers for popular smartphone and tablet operating systems. Some have two or three browsers, while others may have many more.

It's important to test on different web browsers for each platform to check for the following "works on one browser, not the other" type of problems:

- Layout and design issues
- Lack of plugin or 3rd party support
- Different performance for loading images or media
- Performance issues from one to another with larger files
- Differing support for JavaScript and HTML5, some features may not be supported at all in one browser
- Gestures and other inputs may be different from one to another

Try to narrow this down by researching popular browsers. Also try to find out the "problem child" browser on the devices you are going to support. For years on PCs, designers and developers have been frustrated trying to support Internet Explorer 6, which is an old, but still popular web browser. It is often a litmus test - if it can work ok on IE 6, then it will work on more modern browsers.

Poor Security

In a rush to get a mobile presence on the web, sometimes developers forget about security. This is a huge topic, but here are some things you can do on your device to check for security.

If the app or site depends on sensitive information that shouldn't be viewed by others, make sure that it is using secure communication protocols. You can find this by looking for a padlock and other visual cues in your web browser. Even if the app just requires a user name and password, make sure that it is encrypted.

You can do this by testing on a PC with a mobile device emulator, or mobile web browser emulator, and a *network protocol analyzer* tool. Start the network protocol analyzer tool, and use the web app through the emulator for a few seconds. Open up the captured traffic and make sure that it is encrypted, and not human readable. (You may need to get some technical help on how to do this, but your friendly neighborhood programmer will likely be glad to help you.)

If you want to test on the device itself, it's more complicated. If you can find a network analyzer tool for your device, you're set. If not, you'll have to get creative. One option is to use a VNC (virtual network computing) program on the device to access a PC that has a network analyzer program on it. Use the web app on the device through the VNC program, and monitor the traffic on the PC.

You may also work with your systems administrator in an

organization to intercept traffic at the wifi source. This is a great test to do with networking tools, because it simulates how an attacker would try to gain access to your sensitive information. Public wifi hotspots are common areas for would-be attackers to gather information. They attach their own program to a wifi beacon in a restaurant or train station, and watch for insecure data with usernames, passwords and other information that they can capture and utilize for nefarious purposes.

Also make sure that people can't bypass internal measures by tampering with a URL, or directory browsing. These are activities that can be done by changing things in the address bar. URL tampering can occur if information is passed through the URL in the address bar of the app, but isn't encrypted. In the worst case, one app I tested had people's government work identification number in plain text in the URL. If you changed your number, you could end up in someone else's account.

With directory browsing, you just tamper with the address to see if it will take you to a part of the web site that should be off limits. With any URL tampering, the app should display an error that does not give a would-be attacker anything they could use to attack further.

There are **MANY** other facets to security, but these are some simple ones you can test on your own to get started.

Crashing Apps

JavaScript or other coding errors in apps or web pages can cause the web browser to crash, or even reset the device itself. Serious errors can occur when too much memory is used, and isn't freed up (memory leaks), or error or other conditions go on forever (infinite loops.) There are a multitude of programming errors that can creep in whenever we code, but with web technologies, they can be hard to spot during development and debugging.

If my web browser shuts down, it might be because it violates usage and the operating system turns it off. This can occur if it is using too many resources. In other cases, it just might be a serious error that the web browser itself can't handle.

The unique combinations that can cause crashes can be tricky to track down, so watch for areas where an app slows down. If you then do something like a gesture frenzy, or submit something to the web site in a network transition, they will frequently crash at this point. Sometimes, they even restart your entire device. Remember to take everything you can into account when this occurs: network type and strength, battery, movement, and specific things you were doing on the device at the time.

Sometimes we get some sort of useful information from error messages, but a lot of times we don't. If you are working with a programmer, take the device to them and show them what happened. They may be able to get some

information from the device itself with their tools. If you are technically-savvy, use tools yourself to see if you can find crash logs or stack traces to help narrow down the source of these errors so that they can be fixed.

Mobile Web Testing Traps to Avoid

When you are testing mobile web apps or sites, make sure you get out and about and try things out in scenarios that your users will. **DO NOT** sit in your office and do all of your testing in ideal conditions, or you'll miss the most important bugs. Here are some common traps to avoid:

- Only focusing on look and feel, not testing usage
- Using emulators and not the real thing
- Only testing with a strong wifi source in an office, not taking different network types, speeds, transitions and errors into account
- Functional testing and verifying features, not using real-world scenarios

Remember, we are still using a native app - the web browser. It has to load up, work on the device and then display and interpret a web experience for each user. Rather than one app at work such as with a native app, think of it as two apps working at the same time. This adds more strain to the device, and more complication.

Most of what I have described earlier in the book applies to mobile web testing, so try out some tours, learn about the technology, and get out there in the real world. You'll find important bugs quickly, just like with native apps, as soon as you expand beyond functional testing.

Mobile Technologies and the Web

In this section, I will briefly describe some technologies to be aware of when testing for the web.

As we looked at earlier, everything that is transmitted on the internet requires a language to communicate as well. Internet traffic uses a particular protocol, or language for transmitting data called TCP/IP (Transmission Control Protocol/Internet Protocol.) This is a combination of two protocols or language that machines connected to the internet can understand and interpret into our languages for us. The world wide web is what we use, which is a system that runs on the internet, and uses a language called HTTP (hyper text transmission protocol.) When you are doing something that requires security, there is an S added, so it reads "HTTPS" with the "S" meaning "secure."" That means that the transmission is encrypted, so it requires both transmission and reception to translate the encrypted message first, then translate it into something we can understand.

There are some other protocols that are used as well, especially for getting data feeds from other systems. They have

cool names like SOAP (Simple Object Access Protocol), XML (eXtensible Markup Language) and JSON (JavaScript Object Notation.) You'll have to talk to the programmers to find out exactly what the app you are testing uses.

The world wide web is an enormous network of computers that are connected to each other. Our mobile devices connect to this system so we can enjoy the same kinds of experiences that we do with our PCs.

Testing web applications and web sites is quite mature, so I'm not going to go into detail here. *Testing Applications on the Web* by Hung Q. Nguyen is an excellent book to read if you are working on a project that uses web application servers.

There are a couple of things to think about though when you are testing your mobile app, especially if there is some sort of web component, or your app is completely web-based. Also see chapter 10 in this book for more information.

Web Servers

When you surf the web, you are connecting to computers that provide access to web sites, or web applications. These are computers that are purpose-built to supply information or an application for people anywhere in the world with an internet connection. They need to be able to quickly serve up web content to sometimes thousands or even millions of people.

In the early days of the web and e-business, popular web sites or web apps were run off of one incredibly powerful computer. However, we learned that it was much more cost effective to use multiple computers, and it is easier to expand computing power when required. This is much more common today, where popular web sites and applications will use a cluster of servers, and purpose built networking and storage equipment. These systems can be quite complex.

Web applications are developed using different technology stacks, and are governed by specialized equipment so that they can be powerful, fast, reliable and secure. I'm not going to go into details on testing web applications, that is a topic that is well covered by many people, and you'll be able to find a lot of information out there on how to test them. However, if your mobile app is a mobile-friendly web app, or requires a web/app server to work, you will also need to test this part of your system in addition to testing on handsets.

There are advantages and drawbacks to owning and operating your own web or application servers.

Pros:

- Control over hardware: you can get purpose-built systems suited for your application's needs
- Control over performance tuning
- Control over security

Cons:

- Expensive to purchase your own hardware
- Expensive to maintain and upgrade your own systems

The Cloud

In recent years, the cloud has become a popular term. It is a type of utility computing where vendors rent or lease out computing space to other companies. They supply the hardware and infrastructure, and you install your software on it and pay a fee for the service. Rather than own and manage your own hardware, you can rely on someone else's investment and expertise.

Pros:

- Lower cost: no hardware cost, less maintenance costs
- Systems are distributed geographically
- Ease of setup - the systems are configured for you

Cons:

- Physical location - if the cloud service is far away from your users, they may encounter poor performance due to long distances, especially with larger media files

- Loss of control over the systems for performance tuning, security and scaling for growth - they may not be ideally suited for your app
- When there are problems in the system, your app can have poor performance and outages
- If another cloud customer taxes a cloud data center with their app (due to problems, or saturating computing power or network activity) your system might suffer
- More difficult to track down intermittent problems because you can't see what is going on in the system

Common App Technologies Compared

Applications can be built for a specific phone operating system, using a development framework, or they can be created using web technology, and accessed through a web browser. Native apps can take advantage of the features that are built-in to mobile phones, including hardware. They also tend to perform more quickly because they are developed specifically for that platform. Unfortunately, if you support multiple operating systems, you have to create a version for each. That adds a lot of extra work and maintenance.

Web applications have one code base, but they can't take advantage of the built-in affordances of the operating system and hardware. They will run on many devices, but still require testing at least a combination of operating system, device type, and there are lots of web browser programs

to support. They also don't perform as well as native apps because they get their data over a network, and aren't running in the native operating system. They work in a web browser application.

Recently, people have created hybrid technology. You create a web app, but it has access to the features and hardware through an API. You get one code base for multiple operating systems and devices, which is much easier to manage, and a richer experience than a web app, but it comes at a performance cost.

Native Apps

Native apps are programs that are written for a particular smartphone/tablet operating system, and distributed through an application store for that OS. They are programmed using specific languages, libraries and features that are supplied by the OS vendor. There are dozens of mobile operating systems and many development frameworks available, but at the time of writing, these two are the most dominant.

iOS (Apple)

Apple's platform for smartphones and tablets is called "iOS" (i Operating System), and they have a development platform called X-Code, and a vibrant development ecosystem for helping developers create apps for the AppStore.

iOS applications are written in a programming language called Objective-C. Recently, other platforms have come on the market that support writing apps in other languages, but they then need to be translated and compiled in the native Objective-C. Most iOS developers use X-Code and Objective C, with various related tools, supplied by Apple.

If you want to test the application using only an emulator, you will need these tools on a PC.

Android (Google)

The Android operating system is written in the popular and powerful Java programming language. Google supplies you with an SDK (Software Development Kit) and you can pick from several toolsets out there for programming Android apps. You need to use something that supports Android, since the platform doesn't use a generic Java implementation, but has a modified version. One popular development framework is the Eclipse IDE (integrated development environment) with a special plugin for Android development. It provides access to all the Android features, and includes an emulator. Again, if you want to test the app using an emulator, you will need to use the development tools on a PC.

Strengths and Weaknesses of Native Apps

Native apps are incredibly popular, but they are facing stiff competition from web technology. Today, a native app

will perform better and provide a better overall mobile experience, but just as we saw with PC vs. web apps in the late '90s, we may see web apps overtake native apps in the long term. It is a fascinating space to watch.

Pros:

- They run the fastest since they are native code and purpose built for each framework
- Take advantage of all the mobile features and affordances available in a device, both software and hardware
- Each development platform has styles and coding guidelines, so well designed apps will look great on their native platform

Cons:

- Multiple code bases in different languages to support
- You have to submit and manage your app to different stores
- Different design guidelines for each platform means it is hard to get a consistent look and feel

Web Apps

Web apps come in two flavors:

- Accessed through a web browser (just like web access on your PC, no app needs to be installed)
- Embedded dynamic websites (a "shell" app you get from a store install that gets its content from a server)

HTML & JavaScript

HTML is a markup language that is the foundation for most of what you see on the web. JavaScript is a foundational language for helping web applications have more user interaction, dynamic look and feel, and to update information. HTML (with CSS, or cascading style sheets)gets you the look, JavaScript helps web apps or web pages come alive for the end user.

Pros:

- Ubiquitous technology that is supported by every modern computer
- Simple technology, with one code base
- New technology advancements such as responsive design that ensure web sites and apps look great on any screen size

Cons:

- Large files for media, JavaScript code and images can slow performance down because they take a lot of time and energy to download and get operational

- JavaScript that is too complicated, or too chatty with the server causes the device to slow down, or end up with contention errors
- Images and ads don't resize well for different screen sizes, especially smaller ones

HTML5

There are limitations to what you can do on the web, and in the old days we used plugin helper apps to supply more dynamic interaction, media and animation or gaming engines. Two popular plugins were the Adobe Flash player and Microsoft Silverlight. These are fading from use since many smartphones and tablets don't support them. Enter HTML5, a new standard on a set of tools that add more dynamic ability to web pages and web apps. This is a hugely popular and growing web tool for the mobile space.

Pros:

- Standard technology that doesn't require anything but a modern web browser
- Rich media and highly interactive functionality and features for animation, media and more user interaction

Cons:

- Files can be quite large, causing slower download speeds and other performance hits

- New technology, which can be a bit unreliable
- Different web browsers support or interpret HTML5 standards in different ways, so it can be difficult to get apps to work the same on all web browsers

Cross Platform & Hybrid Apps

A recent, popular development toolset has emerged that allows developers to create an app with one code base, and then compile the code onto different native formats.

A more popular toolset are the multiple phone web-based app frameworks, which are development tools that create web apps that may have some hybrid capabilities to access built-in features.

Many are created as an application which is essentially an icon and application that gets all of its content and dynamic interaction from web technology. It's like a custom web site for your phone. They may use HTML and JavaScript and related technologies, or the popular HTML5 tools.

Hybrid apps on the other hand will use a tool for the user interface using web technology, but for each OS they support, they will have an API that also allows for interaction with sensors and other built-in features.

Pros:

- One main codebase

- Ubiquity of the web coupled with native feature access for a richer experience than a web app

Cons:

- Difficult to get a consistent look and feel across platforms (OS version coupled with a web browser)
- Poorer performance than native apps (network, has to run in a web browser program so takes up more resources)

Tracy's Thoughts

Tracy: Web applications require a different outlook when testing compared to native apps. Testing how web apps or web sites work under different wireless conditions is extremely important. If wireless access to the web is slow, unreliable, or changes as you are moving, that can cause problems because web sites and apps depend on good connections to work properly. Since web sites and apps were developed on PCs *for* PCs, they often require far too much inputting and gesturing, and can be very awkward on mobile devices. They also depend on web browsers, which means that there is a lot of complexity so things can go wrong, or they can be very slow and tedious.

Concluding Thoughts

There is a lot of information in this chapter. Don't worry if you need to check back often until you feel like you own the content. It's important to understand what I have shared here, but take your time to really learn it. Make sure you get up and move around as you explore this technology, and try to think about what is happening in the device as well as outside of it whenever you are testing. Mobile experiences on the web have a lot of complexity and dependencies to work that our PC cousins no longer worry about, so don't be fooled! A simple web site might actually be very complex to test properly.

Chapter 10: Don't Forget about Performance and Security!

> "An ounce of performance is worth pounds of promises." - Mae West

I've mentioned a few different approaches to testing mobile apps so far in this book, but I haven't covered everything. In this chapter, I'm going to introduce you to performance and security testing. Each of these topics on their own would be better covered by their own books, but I decided to provide you with a bit of an introduction to help get you started.

Performance Testing

Normally, when I do performance testing work for a company, I test to see how their server or back-end system can handle various amounts of load and whether their client applications meet acceptable targets under different conditions. It can be complex. I have to create special programs that simulate a certain number of users (usually for a web application), execute these tests to get a good set

of data and then apply basic statistical analysis on the data to pinpoint problems.

Once the data shows problem areas, we focus our testing simulation and observation on the problem spots. We have to rerun tests many times to get a proper sample set so that we can analyze it properly. It's sometimes thought of as a bit of a *black art*, but I had a lot of help when I started out from these performance testing heavyweights: Scott Barber[36], Mike Kelly[37] and Ben Simo[38]. I encourage you to look into and study their public work to find out more about performance testing.

With performance testing, we are primarily concerned about speed and accuracy. Does an application perform actions in a way that feels slick and quick, or do we feel like we have to wait for it? Is it always quick, or are there certain times or certain conditions that cause it to slow down?

As Scott Barber taught me when I was starting out in performance testing, you look the effects that different aspects of the system can have on the experience of individual users. Scott breaks it down like this:

- **The application itself**: Complex apps can perform more slowly than simple ones.
- **The device the application runs on**: Different kinds of hardware have different performance strengths.

[36] http://www.perftestplus.com
[37] http://michaeldkelly.com/
[38] http://www.questioningsoftware.com/

- **The network system and related technologies**: Secure vs. insecure communication, networking speed, network providers and related technologies are important to understand.
- **Third-party content providers**: Advertising, data feeds, analytics and other programs our applications depend on may have poorer performance than we do.

These are all areas to analyze when performance testing. But in this section, I am only going to talk about the performance testing we can do with our own devices. We'll look at applications and factors to be aware of when testing performance.

Mobile Devices Perform Differently than PCs

Don't assume that a mobile device will have the same or similar performance as a laptop or other PC. Marketing specs such as CPU clock speed, multi-processors, RAM and others can sound identical, but the architectures are very different. Mobile devices have more physical constraints—especially their smaller size—that impact their performance. If you measure the performance of a laptop on WiFi with a similarly powered mobile device, the laptop will outperform the mobile device because it is using different components and architecture. Furthermore, free disk space is often used to help improve performance, so a

device that is full of songs, apps, videos and games will not perform as well as a one with more free disk space.

Benchmarks

Smartphones and tablets are measured by different types of programs and services to help consumers determine which device is the fastest and best at performance. However, it can be technical and difficult to understand. To get our minds ready for this, I'm going to talk about cars for a moment.

I'm a bit of a car nut. I have a vintage car, a 1955 Pontiac that is highly customized. The engine and transmission is from a 1967 full-sized Pontiac. It is much more powerful than the original engine, because it was developed during the muscle car era.

Here are some technical details: the original engine was a 287-cubic-inch (4.8 liters) V8. That means it has eight cylinders and was quite large and powerful for its day. Power was rated at around 180 horsepower. Later on, this was replaced with an updated version of this engine from 1969, which has 400 cubic inches of displacement (6.6 liters). It is much more powerful, with about 320 horsepower.

Some of these terms and numbers may not mean much to you. They appeal to car fanatics who like to one-up each other, or to racers and collectors. Cars are marketed with some of these facts and figures, but their claims mean a lot more when we understand how they were measured. To

find out how one car stacks up to another when it comes to user experience and performance, the real tests are done by third parties who benchmark the cars.

It is difficult to measure these benchmarks, so strict rules govern how they are completed to avoid wildly different results from test to test. Also, your figures can look more positive if you measure in different ways and only publicize results from one test but not another. (Gasp! Marketing departments only talk about metrics that put their product in the most positive light? Shocking, I know.)

Furthermore, things like horsepower ratings, gear ratios and technical details don't mean much until someone experiences them in the actual product. To provide something that we, the non-technical consumers, care about, other companies perform tests that are more suited to user experience. Here are some common ones for cars:

- **0-60 ("Zero to Sixty")**: The time it takes to go from a stop (0 miles per hour) to 60 miles per hour. Anything under five seconds is incredibly fast—twice as fast as a regular commuter or family car. The less time it takes, the better the performance.
- **Stopping Distance**: How long does it take to stop the vehicle at full braking from a certain speed, such as 60 miles per hour? The less distance, the better (and safer).
- **G-force (gravitational force) Tests**: This measures how much a car moves during hard cornering, stopping and starting, etc. It helps determine how well a

car handles, because a fast car may only work well in straight lines, not on twisty roads. The higher the number, the better the vehicle can handle extreme forces—for example, it can go around corners faster.

- **Fuel Consumption**: This is something many of us take into serious account when we buy a family or commuter car. Again, a lower score tends to be better.

In a sports car, I care about 0-60, G-force and stopping distances the most. If I can afford the vehicle, I can probably afford the higher fuel costs that gas guzzlers cause. In a family car, I care a lot about fuel consumption, stopping distance and safety ratings (which are a whole other family of measurements). Note that no one vehicle solves all problems, satisfies all needs or has all our desired features. There are tradeoffs.

It's the same with mobile devices. Manufacturers publish technical details, such as the following:

- **CPU (central processing unit) clock speed**: This is the "brain" of the computer. It manages all the various tasks it needs to perform. Each tick of the clock will represent many actions, so the faster the clock speed, the faster the processor.
- **GPU (graphics processing unit) speed**: Another brain, but solely in charge of graphics and animation.

- **Hardware technology**: Like the V8 multi-cylinder engine, devices with more GPU and CPU processors are more powerful.
- **I/O (inputs and outputs)**: The speed with which the device communicates with systems outside of it.
- **Memory and storage**: The more space, the more you can process at one time and the more programs and files you can have on your device.
- **Cellular and network speeds**: How fast can your device interact with communication and Internet systems?
- **Web browser performance**: Downloads, uploads and web page rendering.
- **Battery life**: How long will the device work for you when you are on the go?

Third-party services try to come up with benchmarks to make things easier for consumers. What do those specs mean for us? Let's simplify this from a testing perspective. Here are some aspects to be aware of:

- **Processing**: How fast are actions executed on the device? If an app has to do something, is it fast or slow?
- **Rendering**: How long does it take to load a web page or draw an image or animation?
- **Download and upload speed**: How long does it take to download or upload files to and from a server?

- **Combined technology**: What effect might sensor integration, multiple open applications, location services like GPS, low battery and other conditions have on our apps' performance?
- **Different network types, strengths and transitions between them**

Simple Performance Testing

First, find a stopwatch so you can time those really slow actions and provide some metrics. If this seems overwhelming, don't worry. To do a good amount effective performance testing from an application perspective (rather than a system perspective), you only need to be *aware* of what I discuss here. You don't have to be a professional benchmark or performance tester to provide great value.

Third, bring your perceptions and observation skills to the table. Pay very close attention to anything that might seem slow in the app. Finally, start paying attention to your emotions. If you feel frustrated or worry that an action you have performed didn't work, that will point to a performance problem. As mobile development expert John Carpenter says, "If you notice a performance lag, it's a bug."

All you need to do to get started is to go through a simple performance tour. Make your way through the app in a careful, systematic fashion, watching for actions or features that are slower than others. Be sure to note them so you can repeat them again. After you have completed a tour, focus

on each area and log slow actions or features as issues or bugs for the developers. For anything that is really slow (as in, it doesn't just feel slow, but it is measurably slow) use a stopwatch to add some metrics to bolster your report.

To increase accuracy and make this easier, you may be able to team up with programmers who can add instrumentation to the program. The program itself could measure performance and be available through log files or other means.

Real Performance vs. Perceived Performance

The always excellent Smashing Magazine[39] frequently brings up this topic. App designers and developers have a lot of resources to draw on to take advantage of the fact that we can be distracted away from or may not notice some performance lags. Both are important to take into account when testing, and perceived performance is a great place to start investigating. If it feels slow, repeat the action, find out what is bothering you, and then let the designers and developers know.

John Carpenter says that up to 80 percent of his performance-related mobile app development and testing work has to do with the user interface. The remaining 20 percent is spent working to reduce services like GPS and background calls to other services. GPS can sap a lot of device performance,

[39] http://www.smashingmagazine.com/

and as a programmer, it can be easy to get carried away and do a lot of work behind the curtain to try to improve the app experience. If too much is going on in the background, it can slow down the experience for the end user.

The remaining area Carpenter focuses on is battery life. Keeping a screen on (i.e., disabling the sleep or auto-darkening functionality) will take up the most battery life, followed by services like GPS, a lot of background processing, network activity like fetching information from the Internet, and finally CPU processing. An app may have a lot of calculations or rendering to undertake and that can have an effect on overall performance and battery life.

For some reason, we have less patience with mobile devices than we do with their PC cousins. Many experts claim that with a web page, anything taking longer than two seconds to load will feel slow and people will not tolerate it. In the gaming world, any delays above hundredths of milliseconds aren't tolerated. That also applies to visualization software, such as medical apps used by radiologists. Users with mobile devices have expectations that are closer to gamers than web surfers.

Carpenter's experience has shown him that testers and users will report any delays on mobile devices greater than 500 milliseconds. That is a short amount of time, and we as testers don't often have the tools to measure to that level of detail. Carpenter just expects testers to let him know what areas of the app feel slow, the actions they undertook and the device specs and environment they are using the app

in. He then runs special tools to find out metrics if he needs them.

The user interface is what we see and interact with as testers, so here are some areas Carpenter has told me to watch for:

- **Application launch**: We have a lot more tolerance here, but anything above five seconds may cause users to get bored, and that will ruin their impression of our app. A lot of activities are undertaken on app launch, so the dev team may need to reduce some of these.
- **Delays during user interaction or input**: If you notice or sense a delay when typing, gesturing, moving or speaking into the device, note it. That's a performance problem. Immediate user feedback on interaction is important. For example, John points out that a drawing app is useless if the line being drawn doesn't follow the virtual pencil.
- **Anything that doesn't appear to be smooth**: Does something feel jumpy or jittery, whether it is when you are looking at the device or interacting with it?
- **A lack of visual cues when processing**: If a user does something with the app and there will be a delay, we need to let the user know that their input is being processed. Apps often use a spinning wheel to show progress, or will display error or warning messages. If an app doesn't provide visual cues,

people will be confused, and might repeat the action (causing more performance issues), or give up.

- **Visualization, animation and gaming activities**: These need to be incredibly smooth and easy on the eyes. They need to feel like real time, both in playback and when a user interacts with them.

If the app feels slow or unresponsive in any of these areas (or anywhere else you notice), write down the exact steps you took to get there and what you noticed, as well as anything related to your device. If it is extremely slow, use a stopwatch to get some metrics. Also, repeat the actions several times when you have discovered a problem area. (Recall from the beginning of this section when I mentioned that performance testers use basic statistical analysis? A one-time event is hard to fix and can be written off as anomaly.)

If you can repeat a problem every time you try it, that's easy to fix. If it is intermittent, then it's important to know how often it crops up—two times out of ten? Seven times out of 50? If it is intermittent, it will be harder to fix, but it might also be a critical issue that programmers need to know about. If it happens once and you don't see it again, that means you haven't repeated it enough times or you aren't testing in the same conditions as you were before.

Final Performance Thoughts

Performance testing for mobile devices could fill up an entire book. I've only scratched the surface here, but you should still be able to find a lot of areas where the team can improve the app's user experience. Remember, mobile app users tend to have very little patience due to the nature of the devices and how they are used, so don't worry about being too harsh. Your app designers and developers have a lot of tools at their disposal to improve performance, so your feedback will be incredibly helpful.

Performance Test Tips

Here are some things to keep in mind:

- **Watch for delays**: If you sense a delay, there's a performance issue that needs to be logged.
- **Repeat actions**: As mentioned above, it helps to have metrics to show how often this occurs. But, also, repeating a problem when it occurs can cause performance to get worse, possibly pointing to other bugs in the app.
- **Watch startup times**: Devs like to jam in as much as they can in here, so let them know when it becomes excessive.
- **Watch when entering data and clicking to dismiss a keyboard**: Sometimes, a lot of actions are executed in the background. So, again, let devs know when it gets excessive.

- **Understand user expectations!** When I was working on medical imaging apps, what I thought was perfectly fine performance was unacceptable to a radiologist. After spending a bit of time with a couple of potential customers, I learned how to performance test the app completely differently.
- **Load up the device to find bottlenecks**: Problems will be much more apparent when a device is working at its limits than when it is operating under ideal conditions.
- **Use older and "problem child" hardware**: This is another time-saving tool. Older, slower hardware or hardware that is known to perform more poorly will display problems more quickly and with more regularity than high-performance hardware.
- **Watch your battery usage**: If the battery dies more quickly, this will point to areas that need to be optimized and other performance problem areas.
- **If you can't stand the heat**: A lot of heat in the device when you are using an app is often an indication of performance problems. If the device gets hot, poor performance should be your first thought!

Security Testing

Security testing is an important testing approach that can involve many different sub-approaches and techniques, including penetration testing (simulating attack types), fuzzing

(using randomly generated data to attack systems), scanning and auditing using both people and tools. I'm not much of a security tester (I look to the expertise of people like Dr. Herbert Thompson), but I have been involved in security testing projects and audits in the past.

People with ill or criminal intentions like to use security weaknesses to get information about organizations and people to try to make money or simply to ruin things just because they can. They may break into a system and try to gain control or simply steal credit card numbers to sell to thieves.

They can do this by intercepting information and listening in on communication (the digital equivalent of listening to someone else's phone conversation by picking up another phone) or through sophisticated methods of attacking and compromising a system. They may trick people into installing software (or "malware") on their machines that secretly records personal information such as banking info, or they may utilize "phishing" to send a communication pretending to be from a relevant authority asking the user to update personal information. This is common with SMS text message phishing scams. They may also install software on the device that is under their control in order to help attack a larger system or to try to get information from the user's contacts.

Many countries now require organizations to take precautions regarding how they store and use personal data. Identity theft is a big problem, and it can be easy to find that

data with electronic systems. Privacy testing is an offshoot of security testing, and you must make sure that how your app secures, stores, uses and destroys personal information fits the local laws of the land.

As Carpenter points out, location and phone details need to be secured and kept private. Any information that can identify a phone or user must be secured. Having angry customers is bad, but having legal problems is even worse.

Mobile test leader Johan Hoberg notes that Microsoft has an excellent overview for application security that can help you get an overall picture for testing: Microsoft Security Development Lifecycle (SDL) – Process Guidance[40].

Some Security Basics

Security testing through an app alone won't do much. Good security testing also requires a lot of work behind the scenes. The entire system that the application uses needs to be analyzed, and the design, code and use of libraries and communication within the guts of the app require careful use, defensive coding and analysis.

A Warning: To do security testing and development properly, you will need to do much more than I describe here. Security testing depends on special tools, approaches and techniques that go beyond the scope of this book.

[40] http://msdn.microsoft.com/library/cc307891.aspx

Password Protection

This is really simple, but sometimes we overlook the obvious. If a user who wants to steal personal information finds your device, could he do so with ease? Is the app—or at least its sensitive files and areas—password protected? What happens if you set your device down? Does the screen time out and force you to login again, or does it keep you logged in? How about if you go to a different app and come back to it? Is it still logged in, or can you bypass the login by sending it to the background, using a different app and then bringing it up again?

Are there controls to make sure passwords aren't too easy to guess? Do they get obscured as you type them in (usually with "*" symbols), or are they plain text and easily seen by someone nearby?

Also, beware of overly helpful messaging such as password prompts on public interfaces. One of my friends told me a story about an application that had very good error messaging. If you tried to guess the username and password, it would tell you whether the username or password was failing. So, if you guessed a username, you could try to break in with a password-generating program.

If you get the username correct, you are halfway to breaking in!

They did some research and found a programmer's name on the app's web page. A web search for that name yielded an email address, so they plugged the username from the

email address into the username field. Instead of "incorrect username," the message now said "incorrect password."

That is overly permissive messaging. It should tell the user either that the username and password combination is incorrect or, simply, "login incorrect."

Smartphones and tablets are easier to lose and easier for people to steal, so you should always assume that the wrong people could get a device with your app on it. Are there enough simple barriers to prevent them from getting access to sensitive information?

Secure Communications

If your app communicates with other apps, devices or a server on the Internet and the data is sensitive or should be private, then that communication should be encrypted. Since we move around a lot, it is easy for would-be attackers or thieves to set up their own software to listen in on free public WiFi locations. If the data (including usernames and passwords) is sent in plain text and is easily read by humans, then an attacker could take over your application and email accounts or, even worse, capture your banking or other personal information.

On some apps I have worked on, the development team indicated secure communication by adding a padlock icon to the app. We tested this using a device emulator on a PC and a network traffic analyzer program. You could also test this on the device itself by using a virtual private network

(or "VPN") app on your device and having it go through a PC with traffic-analyzing software on it.

Encrypted communication isn't perfect, but it is a huge deterrent for most would-be attackers. In many cases, they will simply move on to a different, unencrypted communication stream, much like a would-be house thief who is deterred by locked windows and doors and a home security system.

Carpenter also points out that third-party tie ins into your app, such as ad networks, logging and analytics tools, may also send privacy information. The app (and development team) is responsible for all that data even if the app isn't sending it.

App Input Attacks

Input fields can be attacked to try to gain control of an application on a device. If a device is stolen or found by a would-be attacker, he may try to use your app for nefarious purposes. To get control of it, he can figure out its error conditions and inject symbols or code into it to get your app to run his malicious code. If you have password security on your app, try the following error modes to see if the app fails the way it normally does when there is an improper username/password combination or if it ends up in a strange state:

- Use special characters and symbols from other languages. There are a lot of them, so be creative!

- Try very long inputs to overflow the processing and error handling. If you really want to stress it, find something 32,768 characters long and paste it in there.

If the app uses files that you can import from the web or another program, create test files that will force error conditions. Try very large files or files with pieces missing. Mix up the formats or the order in which you enter inputs. Do anything you can to mutate inputs (change them repeatedly), and then see how the app handles the error. If it uses a typical error message, that's fine. If it spits out a lot of things that you don't understand, it is probably revealing parts of itself that an attacker shouldn't know about. What might look like a bunch of gibberish to you might be just the info an attacker needs to input a file to take over the app and use it for nefarious purposes.

Technical Analysis

Mobile devices are somewhat different from web applications, because they depend more on the device for user experience. The may save more files locally and may use features within the device. An attacker can utilize a lot of things we can't see in the user interface. To dig more deeply, the entire development team will need to perform an analysis to discover weak points in the app and how to harden up defenses through code, better third-party library utilization, secure and safe data and file storage on the

device and secure communication with other devices and systems.

Attackers use special tools to analyze devices and break into them, and security experts use similar tools to help prevent this kind of problem. To learn more, check out OWASP's Mobile Security Project[41] and the book *Testing Web Security*.

Other Approaches

There are many other important testing approaches that we haven't touched on. I will briefly discuss two of them, accessibility and localization, because they can be incredibly important for mobile apps.

Accessibility

Many mobile devices have excellent built-in support for accessibility, usually for people who have vision problems. Devices have options for screen readers (a program that reads out the text on a screen), voice control (to limit typing and gesture inputs) and alternate inputs and gestures for easier use for sight-impaired people.

Apps do not come out of the box with accessibility support, so development frameworks for operating systems have guidelines for accessible app development. Make sure that you read those when you test.

[41] https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

At its simplest, accessibility testing involves turning on they device's accessibility options and trying the app. Can you launch it with voice control? Does the screen reader make sense? Does it handle other input tools?

Some development frameworks also have accessibility testing tools, so explore that option with your programmer friends.

This is an important factor for many apps that should not be overlooked. My challenge to you is to research and apply to accessibility testing the same approaches we have talked about for other testing techniques. Create an accessibility tour and share it with the rest of us!

Localization

When an app needs to work in a different language, people often assume that all you need to do is translate the words to the other language and verify that the translations work. Unfortunately, testing the translations will only make a small dent into what you really need to test.

For English-speaking developers, any time a non-English alphabet character is used, it can cause failures in various ways. It can fail when it tries to analyze or compute something and the code runs into a character it doesn't recognize. It can fail to display a translation properly on a screen, or it can fail to recognize when you type in different characters. Saving to a file or database can fail as soon as one of these characters is encountered. Also, calculations

can fail on money, time, dates and other numbers because their input format might be different in a different language.

Here are some areas to watch for when testing in other languages:

- **Language**
  - Translations are correct
  - Culture and context is observed, not just technically correct words
  - Time, date and measurement formats
- **Entering special characters**
  - Using a different language than the device is defaulted to
  - Special key combinations will work in a different language
  - Gestures fit within the language constructs and practices
- **Processing and display**
  - Installation (often writes temporary files)
  - Showing characters correctly on the screen
  - Utilizing special characters in calculations or saving and displaying them again
- **Persistence**
  - Saving to a database
  - Saving to a file
  - Logging issues or errors correctly
- **Printing**

  - Looks the same as what you see on the screen
  - Special characters print properly
- **Client or server issues**
  - Special characters are supported on the device but not by the server, or vice-versa

Some special characters are accents (like the accent aigu in French - é), ligature characters made up of two characters put together (æ), ring characters that appear in Scandinavian languages (å), umlauts (ü) that occur in several languages and other languages that use completely different alphabets and symbols. Many parts of the world have different symbols for language alphabets, such as Asian languages like Chinese, Korean and Japanese; Cyrillic in parts of Europe and Russia; Arabic in many parts of the Middle East; Hebrew; and Urdu and Hindi in India. There are many variations of these languages in different countries, and computer programs require special libraries and approaches to handle them.

Duration Testing

Reviewer and mobile testing leader Johan Hoberg reminded me of one of my favorite testing techniques, duration testing. This involves testing the software over longer periods of time. It's amazing how many problems will manifest themselves only after an app has been working for a while.

Tracy's Thoughts

Tracy: Regarding performance, I like the idea of using a stopwatch, since time can seem different than it actually is when you're impatient, waiting for an app to load. If you test with a stopwatch, you have a concrete time benchmarks to bring credibility to your bug reports, and to help create better targets for programmers to shoot for to improve performance.

Reading about security was interesting. I never thought about encrypting the information sent between the mobile device D> and the server, so that is a new area to learn for testing for me. Since I speak more than one language, testing characters from other languages is important because you don't want to alienate anyone who uses your app, even if they D> speak a different language than what the app was developed in. What I didn't know before was that the language and symbols could also cause a problem with hackers or thieves using error codes to their advantage to disrupt or take hijack a device, system or user. Making sure your app can handle different characters in different languages is much more important than just annoying people, it could be a security threat to your users if not handled correctly.

Concluding Thoughts

I added this chapter as a reminder that you probably aren't done testing if you've only used some of the tours and "I SLICED UP FUN." Studying other techniques and approaches always helps me to expand my views and analyze whether we have adequate coverage on a particular project. I encourage you to do the same.

Chapter 11: Final Thoughts

So, you made it this far. Congratulations!

I hope you've learned from my thoughts and experiences. I've tried to share what I have found useful in my own work and, even though this book seems long, we've really just scratched the surface! New versions of mobile technology are coming out all the time, and in a few years, it will be amazing to see where we have taken the technology. The tools and ideas I have shared here are meant to get you started. How you implement them on your projects is up to you.

Mobile technology is constantly changing, and one day this material will seem outdated and quaint. Your approach for adapting, learning and changing your approach to meet new challenges is your own. My challenge for you is to use this book as a springboard. Develop your own approach to learning about mobile technology and figuring out how to exploit new features and devices. Be an amazing tester, no matter what your role is.

Our time is precious, and we are under more pressure to deliver more work in shorter time periods. Use this book to

kick start your thinking. Look for shortcuts to get as much coverage as you can in short bursts.

Dimensions of Awareness for Mobile Testing

As you test on mobile devices, whether you are testing apps, mobile web or a hybrid, you need to be aware of multiple things as you test. It's different from testing on a PC where we don't pay attention to the things around us, or worry about the technology inside the machine that much. With mobile, it is vital to be aware of different things that could have an effect on what we are testing. Mobile devices use sensor-based computing, so you need to be aware of what the sensors are in your device, and what they are doing at any given time.

Network Connectivity

Many mobile applications are dependent on wireless network connectivity to work. In the case of web-based technology (either a thin client on a device or access via a web site), basic functionality is completely dependent on network connectivity. We take this for granted on PCs, but mobile devices can have different network conditions when people are on the move. Always be aware of what technology your device is using, and compare the performance of your app with others that require network connectivity. If

other apps are having problems with network conditions, that is a good time to take note of what your app is doing.

Movement

Mobility is all about moving and movement: away from home or the office, or within a building. Moving while walking, riding a bicycle, or from within a vehicle. It also means we can move from one physical location to another geographically, and we still have access to the things we need to be productive.

The devices have movement sensors (usually an accelerometer which detects any movement, a gyroscope that detects 360 motion, and a magnetometer that detects your position relative to the magnetic poles.) There are also sensors that detect touch inputs, usually a combination of pressure, and capacitance (we give off a slight electrical charge, and the device uses this to get very accurate representations of what we are doing with our fingers.) On native applications, movement of the device such as a tilt, a shake, or panning the device can be used as alternate inputs as well as touch screen gestures.

Movement has an enormous impact on the utility and usability of apps. Can you see the app properly when you are moving? Can you interact with it and still do required gestures while moving? What happens if you rotate the device from portrait to landscape and back while using the app? What happens if you inadvertently gesture and do the wrong sorts of inputs? Can the app handle that?

What happens if the app is active as you walk up and down stairs, or trip or stumble? All of these areas are important to incorporate into testing because that is what happens with the devices in the real world, and can cause difficult problems, especially if something that seems simple to us (oops, I tripped) is really complex for the app and device to handle. (Multiple inputs at once, multiple sensors being utilized, app is downloading or processing something, etc.)

Location

Location services are quite complex, and can be used to determine your current location to provide information to a system to save the end user having to do it themselves, to provide custom content based on where the person is located, or to trigger other events. There are dependencies on wireless connectivity, and several technologies can be used at any given time. GPS is the best known, and it has its own wireless technology, and it can have interference from tall buildings, steel and even cloud cover. Wifi routers and cellular towers can also be used to help determine location. Movement sensors can determine even more accuracy such as the direction that a user is facing (or the direction of their device is pointed at.)

When you are testing, it's important to make sure whether location services are being utilized, and whether they are being used correctly, but also to be aware of what is going on in your current location.

Environment

Where the application or device is being used has an effect. What if the person using the device is in an area of the building that has weak wifi, or interference so it won't work? What if they are in an area where there is weak, poor or no cellular connectivity? It's important to understand what technology your device is using, and whether the network connection is working as it usually does. Some areas are particularly difficult – areas away from urban centers, inside tunnels or elevators, or in areas where there is a lot of radiation from other equipment such as a hospital, airport or vehicle service center.

Weather has an effect on how an app will work. Thunderstorms have a lot of electrical activity which can effect cellular networks due to interference with the wireless waves. Cloud cover can impact the reliability of GPS. Snow or cool weather can cause your fingers to get cold, which means you will have trouble interacting with small objects on a screen. Heat can cause the device to go into an optimization mode as it tries to shut things off and slow itself down so that it doesn't overheat.

The devices have light sensors that help optimize their use. Under bright lights, the screen will dim, and in low light, the screen will brighten. Some app designs using very light or very dark colors may not be visible in natural lights, especially in the dark or in bright lights.

Device States

The device itself has different operational modes. From a testing perspective, it's important to understand optimization modes in particular. When the device is in low battery mode, it will try to shut things off and reduce energy consumption to preserve the battery. When the device gets hot, it will try various things to cool itself down since it doesn't have a fan or any cooling system. It will slow down the clock on a CPU, free up memory and turn off certain affordances. If it gets too hot, it will shut itself off to try to prevent damage.

There is also a proximity sensor which can tell if you are in front of the device or not. The most battery intensive activity is lighting up the screen, so the device will try to shut the screen down if they aren't interacted with. If you haven't interacted with the screen, and you are away from the device, the proximity sensor combined with a lack of interaction signals to the device that it is ok to turn off the screen.

The End Users Expectations and Emotions

What are the reasons they are using the device? How do they intuitively respond to, and interact with a physical device that depends on movement to use? Do they make slow, measured inputs, or are they fast and unpredictable? What are their motivations i.e. What problems does the app

solve for them, and what are their fears i.e. What happens to them when things go wrong?

Put yourself in the user's place. Also consider emotions: frustration, anger, feeling rushed, feeling confused. That can cause physical changes in a person's body, which can impact how they interact with the device. Different emotions may cause the same person to interact with it differently, which can have an effect on how the app responds to inputs and movement.

Furthermore, if you are frustrated or in a rush, you have far less patience for poor performance, or for entering in a lot of data or information.

The Seven Deadly Sins of Mobile Apps

To close the book, here is a simple summary I created to help you think of different areas in which an app can fail. This is another mind hack or mnemonic you can use to organize your thoughts and analyze an application quickly.

- **Lust**: The app advertises that it can do a lot more than it actually can. It leaves you feeling unfulfilled and wanting more.
- **Gluttony**: The app uses far too many resources. It eats up your memory, downloads large images and other files, slows down your device, eats up your data plan and kills your battery.

- **Greed**: The app assumes you have a strong network connection and can't handle poor or weak connections or transitions.
- **Sloth**: The app performs very poorly. It is far too slow.
- **Wrath**: The app doesn't play well with other apps. It has special settings that override your defaults, and it doesn't allow other services to work well with it.
- **Envy**: The app is a copycat. It is too close to other available apps out there that you would prefer to use instead.
- **Pride**: The app is difficult to use, expecting users to adapt to its way instead of helping you be more effective. You are subordinate to it, you have trouble using it, and you end up feeling stupid. This is most likely due to designers and developers assuming they know better than the users and ignoring valuable feedback and usability bugs.

Thanks to Shannon Hale and Jared Quinert for their help with this list.

Tracy's Final Thoughts

Tracy: This book sets you up for success. As a new mobile tester, reading this book has given me a toolkit and reference guide. Without it, I don't think I would have done very well as a newbie. The book gives you all the tools you need to test mobile apps successfully. It's also easy to read and understand. Jonathan takes what seem like complex and overwhelming tasks and distills them down in a way so they seem like common sense. It's all laid out there for you in an easy to read format.

Now the rest is up to you - take the ideas and make them your own, just like I did.

Further Resources

If you would like to learn how to become a better tester, one of the best resources on the planet is Cem Kaner and Becky Fiedler's Black Box Software Testing Course[42]. The course is free for self study. Work through it at your own pace, and you will have an amazing grounding in software testing.

Good luck, and happy testing!

[42] http://www.testingeducation.org/BBST/

Appendix 1: Tools and Considerations

"We shape our tools, and thereafter our tools shape us." - Marshall McLuhan

You will need some basic tools to get started. Here are a few different options you will need to consider. I've also included other considerations and useful things to be aware of when testing that aren't really tools, per se, but are vital for certain kinds of testing.

Mobile Devices

It might seem obvious, but you'll need at least one smartphone or tablet to do real-world testing. Simulators or emulators won't help for much beyond basic functional testing. If you can, find out what target platform(s) the development team is releasing software for and see if what you have matches up. If not, you may need to procure devices for real-world testing. In this book, we focus on smartphones and tablets, so you'll need at least one of those to get any further than this sentence.

Network Connectivity

If the applications you are testing require a connection to the web or to a corporate network, you will need to have wireless access. You usually get this for free using Wi-Fi on most devices, but for the times when you are away from Wi-Fi beacons, you will also need to get cellular data access through a communications provider's wireless broadband plan.

Wi-Fi

If you don't have a router or wireless access point that provides Wi-Fi for you at home or at work, scout your neighborhood for shops that provide free Wi-Fi access. Many coffee shops, restaurants and other services provide free Wi-Fi for customers.

Cellular Network Access

Data Plan

This may be optional, but many mobile applications depend on connecting to servers and other computer systems and transmitting and receiving data from them. If the application you are testing requires connection to other systems or computers, you will need a connection to the Internet. When you aren't using Wi-Fi, your telecommunications provider will provide access to its wireless broadband networks, usually for a monthly fee.

Deadspots

It's important to know where there are deadspots around you so you can use them when testing. A deadspot occurs when you have no wireless connectivity. Steel structures interfere with wireless waves, and so do microwaves when they are operating. There can also be gaps in coverage where wireless infrastructure doesn't work.

You can use a microwave shell as a deadspot, or as a faraday cage when it isn't operating. Simply place the device inside, and attempt an action that requires an Internet or network connection. *Do not close the door and turn the microwave on.*

Mobile Phone Communication Plan

Mobile phones are devices with a computer and a phone (radio) built in. How the communication features and the computer aspects interact with each other is important to test. Make sure your test smartphone can send and receive voice, SMS texts, other messages, etc.

Space to Move

Mobile devices contain movement sensors that application developers can utilize for application development, and interaction occurs through touch screens. To test adequately, you will need to be able to move the device around in order

to engage sensors, to try out how an application works when walking, and to see how the application works when network conditions change.

Note-taking Tools

There are a lot of simple electronic tools to take notes with, but if you are traveling around while testing with your mobile device, I recommend something simple like a notebook. I use a Moleskine notebook and checklists that are printed out, as well as reminder applications on mobile devices like Todo list apps.

Your Brain and Imagination

Mobile devices are used in countless contexts, environments, situations in different weather and for different reasons. Users will also be in different emotional states when they use these applications. On the devices themselves, sensors, network, memory, battery strength and other conditions will come together to create either a great user experience or, when they go wrong, strange crashes or bugs. You're going to need to observe how people use applications and channel that into testing your application. Couple that knowledge with what you will learn about these devices in the book, and you can generate a lot of interesting test ideas.

You will also need to be creative to deal with intermittent problems. There are a lot of areas where these features can combine under specific and unique conditions to create failures that can be difficult to repeat at first. With a little knowledge and a little creativity, these can be tracked down much more easily.

Testing Equipment

Setting up space with dedicated equipment and storage is incredibly important for mobile testing efforts.

- A PC for using productivity tools
- PC-based testing tools such as emulators or simulators
- PC-based tools for configuration
  - Apple has a Configurator tool for managing multiple devices and the iPhone Configuration Utility for single devices: https://www.apple.com/support/iphon

Device Emulators

Development frameworks provide their own device emulators as part of their development tools:

- Apple iOS: XCode

[43] https://www.apple.com/support/iphone/enterprise/

- Android: SDK (Software Development Kit)
- Windows Phone: Windows Phone SDK
- RIM: BlackBerry Developer Tools

You need to download and install the development framework and tools on a machine to be able to use them.

Development kit emulators can help you check for look-and-feel issues and really obvious problems on several device types. You can also use them to force error conditions such as low memory.

Network Traffic Emulators

There are now traffic emulators to help simulate different cell speeds and error conditions. These are useful to use on a PC with an emulator to do some basic network testing or to simulate error conditions that are difficult for you to do with a physical network. One example is the Apple Network Link Conditioner, which can be installed with XCode developer tools.

The AT&T Application Resource Optimizer tool is a useful tool for network optimization and other important factors.

Bug Logging

When you find important issues, it's best to record them in a way that they won't get lost. You can start simple and get more sophisticated if you are just starting out:

- Email, texts and instant messages
- Bug-tracking system and productivity or ticketing system

Communication Tools

Be sure to utilize the technology that is built in to the devices to help you.

- Email
- Bug-tracking software
- Instant Messaging
- Text messages
- Pictures and video

Platform Testing Tools

You won't be able to test all the combinations on physical hardware, so supplement some of your basic testing, or "smoke testing," by considering the following tools and services.

Crowdsourcing services

People sign up to these services to allow for the use of their devices for testing. This testing isn't highly skilled, so be careful what you use the services for. However, with a bit of careful planning and strategy, you can get a lot out of these services.

Remote Handset Device Access Services

Some organizations have purchased devices and set them up for remote usage over the web, so that you can install an application and perform basic functions on the devices remotely. Again, these are basic tests, but they can be helpful.

There are free services as well as pay services from companies so shop around.

Utilize Your Own Connections

Take advantage of the devices that your friends have, and ask them to help you out for some basic tests.

In real life, you can set up a meeting with friends to do an "applications test over appetizers" at a coffee shop or restaurant. Use social media to find out which friends have which devices, and gather a small group who have devices that you don't have access to. Spend time on some basic tests so that your friends can easily repeated them in a social setting. Pay for food and drinks, manage some testing and find bugs while having fun with your friends. It's inexpensive and can be really helpful.

If you like to organize events locally, use tools to set up a local meetup through a local applications developer or marketing group to help you test on other platforms with other people.

In the virtual world, coordinate electronically with friends in other countries through social media. Get them to try out applications and let you know how they work on their devices in their countries.

If you are comfortable with working with strangers, also consider setting up a virtual test bash or flashmob by getting interested testers in other parts of the world to join up. Search for weekend testing activities online, and see if you can use an existing group to facilitate some testing.

Management Tools

You will want to use tools to get stacktraces off of devices for bug reports and to manage things like SSL certificates or other configuration activities.

For Apple, you can use the iPhone Configuration Utility to manage one device and for getting stack traces. You can use Apple's Configurator for multiple devices in a test lab. Research similar tools on other platforms, they are incredibly useful.

Appendix 2: Mobile Content Snack Tests

"A quick test is a cheap test that has some value but requires little preparation, knowledge, or time to perform." - Cem Kaner & James Bach

To get started with an application, you can try out what I call "content snack tests" for mobile, which are a variation of quick tests. These are simple actions that can help you warm up your testing brain, and in many cases, they yield bugs.

- **Portrait vs. landscape switch**: Start using the application in one view, rotate the device and repeat. Variation: try rotating several times in rapid succession and then use the application.
- **Gesture frenzy**: Use a touch input on the screen on every square millimeter of the screen. Try different gestures: tap frenzy, swiping, press, scrolling, pinch and expand. Be sure to do them over and over in rapid succession.
- **Try it standing up**: Work with the application while seated, then stand up. Does the change in position cause you to see anything differently? Is the application still easy to use?

- **Change your mind**: Start working through the application, and then try to get back a screen. Try to get to the beginning. Try to reset it to a default state. Can you go back?
- **Shake it**: Try an accidental shake while entering data, interacting with the application or while it is trying to connect to the Internet.
- **Walk and stop**: Try the application while you are on the move. If the application uses location services, see what effect stopping and starting while using the application has on your location accuracy.
- **Stop using the device and come back**: Work with the application, then put your device down. Come back to it a few minutes later and see if you can pick up where you left off.
- **Alternate obsessively**: Work with the application and, while it is processing and you are waiting for it to do something, go to other applications on your device, such as email, social media, etc. and come back. Switch back and forth between your application and other applications.
- **Do something while a page loads**:
  - Gesture frenzy
  - Constant reload (often a swipe downwards)
  - Sensor overload: Move the device around
  - Move and walk around
  - Cover and uncover light and proximity sensors on the front of the device
  - Combine all at once!

Try it on the move: Can you see each screen and each element on each screen while you are walking or moving in a vehicle? Can you gesture and do all of the required inputs with the app while moving? Now, can you do all of those things with ease while moving or do you need to stop moving at times to either see or interact with the app or web site?

These are some examples that you can use to get your brain going and discover something interesting about the application. They may seem like party tricks, but they come in handy. I use them to get started or when I feel stuck and can't think of anything else to test. Feel free to discover and create your own, this is just the beginning.

For more advanced ideas to quickly try out different testing perspectives, see the chapter on testing tours.

About the Author

Jonathan Kohl

A thought leader in mobile application development, Jonathan Kohl has been a pioneer in applying design & business analysis, project management and testing on mobile application projects for smartphones and tablets. Jonathan is an internationally recognized consultant and technical leader. He helps companies define and implement their ideas into products, works with leaders helping them define and implement their strategic vision and helps technical teams create great client experiences.

With over 15 years of experience in the software industry, Jonathan has consulted independently since 2004, providing consulting, training and hands-on project work. He has published over 40 articles and has contributed to three

books, including Peggy Ann Salz's *The Everything Guide to Mobile Apps*.

Jonathan is founder and principal consultant for Kohl Concepts Inc. He lives in Calgary, Alberta, Canada with his wife Elizabeth. Contact him at www.kohl.ca[44] or email jonathan@kohl.ca[45].

[44] http://www.kohl.ca
[45] mailto:jonathan@kohl.ca