

Don't Panic

MOBILE DEVELOPER'S GUIDE TO THE GALAXY



published by:



Enough Software GmbH + Co. KG
Stavendamm 22
28195 Bremen
Germany
www.enough.de

15th Edition February 2015

This Developer Guide is licensed under the
Creative Commons Some Rights Reserved License.

Please send your feedback,
questions or sponsorship requests to:
mdgg@enough.de

Follow us on Twitter: *@MobileDevGuide*

Art Direction and Design by
Cornelius Kwietniak
(Enough Software)

Editors:

Marco Tabor
(Enough Software)

Julian Harty
Chris Ward

www.mobiledevelopersguide.com

Mobile Developer's Guide

Contents

The background of the page is decorated with a space theme. It features several brown, irregular shapes representing asteroids or planets scattered across the upper half. There are three yellow, five-pointed stars. In the bottom right corner, a green, cartoonish alien with large white eyes and a small smile is peeking over a purple, rounded horizon line. The overall color palette includes shades of purple, pink, blue, brown, yellow, and green.

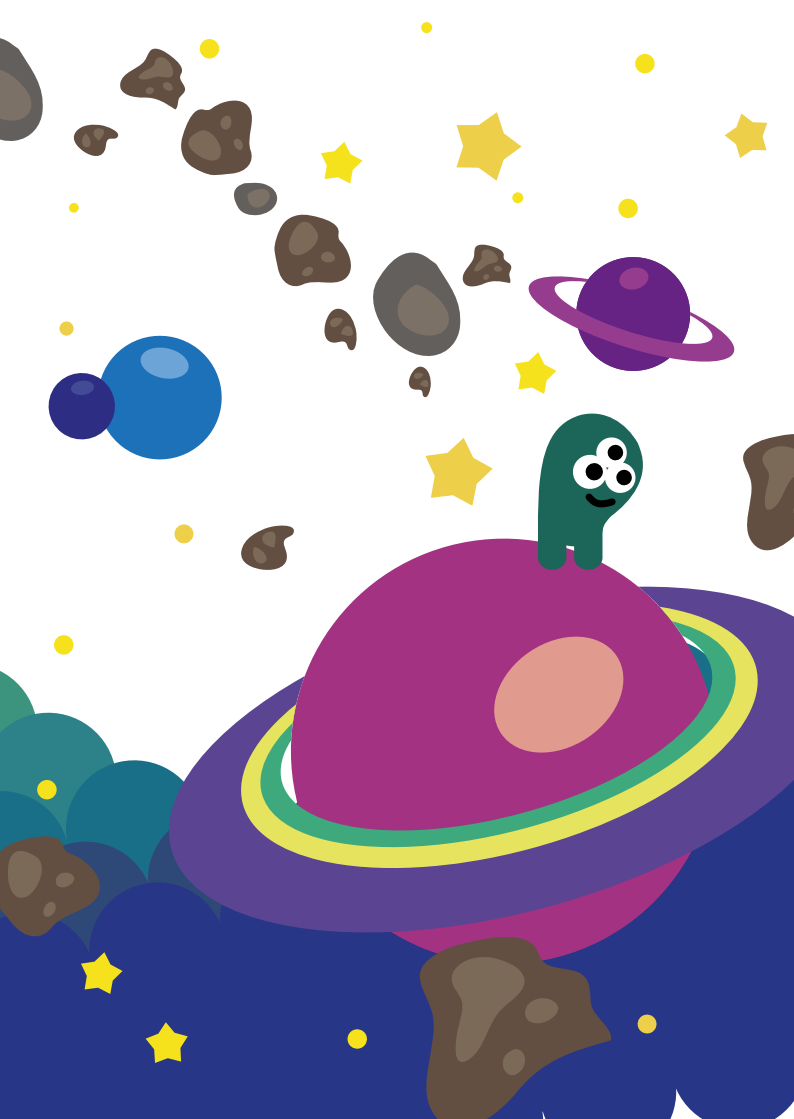
- 1 Prologue
- 4 The Galaxy of Mobile: An Introduction
by Robert Virkus & Marco Tabor
- 16 From Idea To Concept
by Sebastian Meyer
- 26 User Experience & Interface Design
by Anna Alfut
- 36 Android
by Andre Schmidt & Tim Messerschmidt
- 52 BlackBerry 10
by Marcus Ross
- 62 Firefox OS
by Marcus Ross
- 68 iOS
by John Gambrell
- 82 Java ME (J2ME)
by Ovidiu Iliescu
- 92 Tizen
by Marco Büttner & Patrick Mortara
- 98 Windows
by Robert Virkus
- 108 Going Cross-Platform
by Robert Virkus & Michael Koch



- 124 Mobile Sites & Web Technologies
by Daniel Kranz
- 138 Enterprise Apps
by Ian Thain & Davoc Bradley
- 146 Mobile Gaming
by Oscar Clark
- 164 Mobile Development & the Internet of Things
by Alex Jonsson
- 172 Programming Smartwatches
by Robert Virkus
- 180 Mobile Analytics
by Julian Harty
- 192 Application Security
by Dean Churchill
- 204 Accessibility
by Sally Cain
- 224 Testing
by Julian Harty & Marc van't Veer
- 246 Monetization
by Michel Shuqair & Carlo Longino
- 262 Epilogue
- 263 About the Authors







Prologue

App development has gone a long way in the last 10 years. When we started Enough Software in 2005, mobile apps were really quite exotic, limited to phones; and distribution was difficult, to say the least. Now we not only target phones as app developers, we have broadened our reach to support tablets, phablets, PCs, smartwatches, TV systems and Automotive systems. Also many Internet of Things (IoT) products come to life thanks to apps.

In this 15th edition, since 2009, we reflect these new opportunities in our new chapters; most notably the IoT and the smartwatch chapters. But also all the other chapters have been extended, improved and sometimes even completely revamped. At this point we would like to thank all of our authors - you are doing a great job!

We also love our sponsors who make it possible to print this booklet: SAP, HP and Microsoft! Please visit their mobile platform offerings at developers.sap.com, developers.hp.com and dev.windows.com!

Please share your excitement with us via twitter [@enoughsoftware](https://twitter.com/enoughsoftware) or via email: mdgg@enough.de. And visit mobiledevelopersguide.com to obtain the electronic edition of this booklet, which is available in several languages.

Robert + Marco / Enough Software
Bremen, February 2015

PS: Want us to help you succeed in the mobile ecosystem? Drop us a line at service@enough.de to learn more about our app coaching services.





The Galaxy of Mobile: An Introduction

Welcome to the world of mobile development, a world where former giants stumble and new stars are seemingly born on a regular basis.

The focus of this book is on developing mobile apps, which encompasses a number of phases including: planning and specification, prototyping and design, implementation, internal testing and deployment, deployment to an app store, discovery by users, installation, use and feedback. Ultimately, we want our users to enjoy using our apps and to give us positive ratings to encourage other users to do likewise.

Keep reading to learn how to develop apps for the major platforms. Should this be the first time that you have considered getting involved, don't delay; the world is moving rapidly towards mobile becoming the predominant form of computing and others will surely overtake you if you wait too long.

While developing mobile apps shares many common features with developing other software, it has specific characteristics. We will cover some of these next.

Topology: Form Factors and Use Patterns

Traditionally we app developers only targeted phones. Then tablets followed, and today our apps can span across a complete range of device types: smartwatch, phone, tablet, PC, TV and automotive. Other form factors might follow. Each form factor poses its own usability challenges; for instance, a tablet demands different navigation to a phone, input on TV systems can be cumbersome, and so on.

Use patterns in an Android app, of course, differ from those on iOS, which also differ from those for Windows Phone apps, et cetera. You should, therefore, refrain from providing an identical experience on all form factors or even all you target are smartphones. Otherwise, you risk delivering a mediocre service to various sections of your target user base.

Star Formation: Creating a Mobile Service

There are several ways to realize a mobile service:

- App
- Website
- SMS, USSD¹ and STK²

App

Apps run directly on the device. You can realize them as native, web-based or hybrid apps.

Native Apps

A native app is programmed in a platform specific language with platform specific APIs. It is typically purchased, downloaded and upgraded through the platform specific central app store. Native apps usually offer the best performance, the deepest integration and the best overall user experience compared to other options. However, native development is often also the most complex development option.

¹ en.wikipedia.org/wiki/USSD

² en.wikipedia.org/wiki/SIM_Application_Toolkit

Web Apps

A web app is based on HTML5, JavaScript and CSS and does not rely on any app store. It is a locally stored mobile site that tries to emulate the look-and-feel of an app.

A famous example of a web app is the Financial Times app, which left the app store in order to keep all subscriber revenue to themselves for the web world; conversely, the web-based Facebook iOS app was revamped into native app in order to dramatically improve its performance and usability. There are several web app frameworks available to build a native wrapper around such apps so that you can publish them in app stores, such as Phonegap³.

Hybrid Apps

A hyped controversy circles around whether native or web apps are the future.

For many mobile app developers, this controversy is no longer relevant as a hybrid approach to app development has become quite common: an app can use native code for enhanced performance and integration of the app with the platform, while using a webview together with HTML5-based content for other parts of the app. Parts of the resulting app behave like a native app, while other parts are powered by web technologies. The web-based part can use Internet connectivity to offer up-to-date content. While this could be viewed as a drawback, the use of web technologies enables developers to revise content and features without the need to submit updates to app stores. The key challenge is to combine the unique capabilities of native and web technologies to create a truly user-friendly and attractive app.

3 www.phonegap.com

Website

A website runs on your server but you can access various phone features on the device with JavaScript, for example to store data locally or to request the current location of the device. In contrast to apps, mobile websites are inherently cross-platform. Historically mobile websites often catered for WebKit based browsers such as the Mobile Safari. Nowadays, WebKit and Chromium are the dominant mobile rendering engines with Internet Explorer's Trident engine and others follow behind in the mobile space. For the sake of an open web, aim to use HTML5 standards as much as possible and refrain from rendering engine-specific code.

SMS, USSD and STK

Simple services can be realized with SMS, USSD or STK. Everyone knows how SMS (Short Message Service) text messaging works and every phone supports SMS, but you need to convince your users to remember textual commands for more complex services. Some operators offer APIs for messaging services that work for WiFi-only devices, such as the network APIs of Deutsche Telekom⁴. In the UK in many towns parking charges can be paid using SMS messages.

USSD (Unstructured Supplementary Service Data) is a GSM protocol used for pushing simple text based menus, the capabilities depend on the carrier and the device. In Sri Lanka, visitors can receive a free SIM card which is registered using USSD menus.

STK (SIM Application Toolkit) enables the implementation of low-level, interactive apps directly on the SIM card of a phone. STK may appear irrelevant when so much focus is on smartphone apps; however, for example, m-pesa is an STK app

⁴ www.developergarden.com/apis

which is transforming life and financial transactions in Kenya and other countries.⁵

The Universe of Mobile Operating Systems

The mobile space is much more diverse than other areas in IT. When you are developing software for personal computers, you basically have 3 operating systems to choose from. When it comes to mobile, there are many more. This book provides an introduction to the mobile operating systems that are currently the most relevant. Be aware, the mobile space changes continuously and at a speed that you will seldom observe in other businesses. We have seen many promising technologies appear and quickly disappear, regardless of how big the companies behind them are, or the historic market relevance of those companies.

So read on; learn how the market is today and then be prepared to track the changes (or make sure you have the latest edition of our guide available).

Quasars: Android and iOS

When people talk about mobile apps, they mainly refer to Android and iOS. Why? When it comes to market share, these two platforms combined dominate the smartphone market with easily 90% in key markets⁶ (see the table below for global numbers). The Developer Economics 2014 research⁷ also shows that iOS and Android are at the top in terms of developer mindshare - that is, the percentage of developers using each

⁵ memeburn.com/2012/03/how-m-pesa-disrupts-entire-economies/

⁶ www.idc.com/getdoc.jsp?containerId=prUS24442013

⁷ DeveloperEconomics.com

platform, irrespective of which platform they consider to be their 'primary'. Android was at the top, with 71% of developers currently working on the platform, followed by iOS with 55%.

Of course this also means: if you are going to use Android or iOS, you will have lots of competition.

Dark Matter: Feature Phone Platforms

While smartphones generally get the most news coverage, some parts of the world still belong to the feature phone universe. Globally one third of all phones sold in Q3 2014 were feature phones⁸, with an install base much higher than that. However, Android is increasingly taken over the low-cost handset market so the future of this platform looks dim.

Even the big players in the feature phone market have to realize this: Nokia plans to completely shut down their feature phone app store during the first half of 2015. It will be "replaced" by Opera⁹.

While you can develop native apps for feature phones when you have close relationship with the vendor, you typically develop apps using Java ME or BREW for these phones.

Magnetar: Windows

Windows has now become the 'third ecosystem'¹⁰ in the smartphone universe, it even sells out iPhone in some regions, such as Italy or Argentina. Windows 8.1 and Windows 8 market share has now surpassed the share of all Mac OS X and even Windows XP versions combined according to Net Applications¹¹.

⁸ gartner.com/newsroom/id/2944819

⁹ blogs.opera.com/news/2014/11/nokia-store-become-opera-mobile-store

¹⁰ kantarworldpanel.com/global/News/news-articles/Apple-iPhone-5S-outsells-5C-three-to-one-in-Great-Britain

¹¹ netmarketshare.com/operating-system-market-share

While Windows 8 and 8.1 were never that popular among customers, it seems possible that Windows 10 will turn this impression around. Windows 10 will power a large variety of devices, starting at embedded systems and wearable devices, supporting phones, tablets, PCs and powering the Xbox One as well.

Super Novas: Sailfish OS, Firefox OS, BlackBerry 10 and Aliyun

Will these platforms become spectacular success stories or doomed chapters of the mobile industry? Nobody knows for sure, but there are mixed messages open for interpretation.

The Finish company Jolla¹² entered the market in Q4 2013 with its Sailfish OS¹³. While still being a niche system, Jolla brought out a steady release of its firmware and has raised over 1.8 million USD via crowdsourcing to realise a Jolla tablet¹⁴ in May 2015.

Firefox OS¹⁵ has not really taken off so far. Originally aimed at low-cost devices it turned out that the overall-experience with a web based system is not optimal on such hardware. Of course hardware continues to improve in 2015, so it will be interesting to experience Firefox OS on cheap but powerful handsets.

BlackBerry 10 refuses to die - and for good reasons as well. As security awareness continues to increase in this connected world, BlackBerry brings some interesting concepts to this table. While decidedly being niche, it seems to be a lucrative and cozy niche.

¹² jolla.com

¹³ sailfishos.org

¹⁴ www.indiegogo.com/projects/jolla-tablet-world-s-first-crowdsourced-tablet

¹⁵ mozilla.org/firefox/os

Aliyun has been released in 2014 on a single device in China with an unknown market share. As no follow up devices, err, followed up, it is safe to assume it being dead from a developer's perspective.

White Dwarfs: Symbian, bada and other dead systems

Some operating systems have been fading away (like Samsung bada), some have stopped with a bang (like WebOS) and some have been super-seeded by new developments (Windows Mobile). Why did some succeed where others failed? In the end it boils down to marketing, developer mindshare, company politics and a big portion of pure luck. It has now become increasingly difficult to compete with the massive ecosystem weights of Android and iOS, a fact that currently seems to be continued into the smartwatch market.

Newborn Stars: Tizen and Ubuntu

Promised for 2014, we still have not seen Ubuntu for Phones powered devices entering the market. The idea is to bring the full power of a PC to the phone. The crowdsourcing¹⁶ effort to fund the Ubuntu Edge phone did not reach its goal, but Canonical is still expected to enter the market in "early 2015"¹⁷.

Tizen¹⁸ has enjoyed quite a success in the smartwatch market (compare our smartwatch chapter), however previously promised mobile phones have been delayed by Samsung. In January 2015 the first Tizen powered smartphone has finally been launched in India, the Z1. Seemingly gently yet continuously pushed forward by Samsung and Intel, Tizen aims to power not only smartwatches and smartphones but also TVs,

¹⁶ indiegogo.com/projects/ubuntu-edge

¹⁷ pcworld.com/article/2861446/ubuntu-phone-launch-delayed-until-early-2015.html

¹⁸ tizen.org

tablets, netbooks and in-vehicle infotainment systems. The fact that we included a dedicated chapter about Tizen in this edition of this guide reflects the fact that we are taking the platform seriously.

Solar System: Smartphone OS Market Shares

When you look at the global smartphone market shares, the picture might look simple:

Platform	Market Share			
	Q3 2014	Q3 2013	Q3 2012	Q3 2011
Android	84.4%	81.2%	74.9%	57.4%
iOS (Apple)	11.7%	12.8%	14.4%	13.8%
Windows Phone	2.9%	3.6%	2.0%	1.2%
BlackBerry	0.5%	1.7%	4.1%	9.6%
Other	0.6%	0.6%	4.5%	18.8%

(Source: idc.com/prodserv/smartphone-os-market-share)

You may agree with the majority of developers that decide spending time on platforms other than Android and iOS is a waste of time. Be assured: It is not that simple. While world-wide smartphone shipments exceeded feature phones¹⁹ for the first time in Q1 2013, feature phones still outsell smartphones in many regions, and Nokia Asha is targeted as the platform for the "next billion mobile phone users"²⁰ and includes porting guides **from** Android.

Also, be aware these are global figures: the regional market share of each platform varies significantly. In a world where localized content is increasing in importance, it is vital to

¹⁹ idc.com/getdoc.jsp?containerId=prUS24085413

²⁰ developer.nokia.com/asha

know the details and characteristics of your target market. For example, China is the largest smartphone market today responsible for more than 40% of worldwide Android shipments in Q3 2013²¹, but Chinese Android handsets are typically based on the Android Open Source Platform (AOSP) and come without the Google Play Store or the Google Mobile Services.

To find out about market share in your target region, check out online resources such as comscore²², StatCounter²³, VisionMobile²⁴, Gartner²⁵ or Kantar Mobile World Panel²⁶.

About Time and Space

As developers, we tend to have a passion for our chosen darlings. However, let us not forget that these technologies are just that - technologies that are relevant at a given time and in a given space, but not more. Yes, flamewars are fun but in retrospect, they are always silly. Hands up those who fought about Atari versus Amiga back in the good ol' 80s! Probably not many of you but, surely, you get the point. Initiatives such as FairPhone²⁷ or IndiePhone²⁸ may prove more important than the OS or vendor of your choice in the future.

21 engadget.com/2013/11/14/android-ios-market-share-gartner-q3-2013/

22 www.comscore.com/Insights/Data-Mine

23 gs.statcounter.com

24 visionmobile.com

25 gartner.com

26 kantarworldpanel.com/global/smartphone-os-market-share

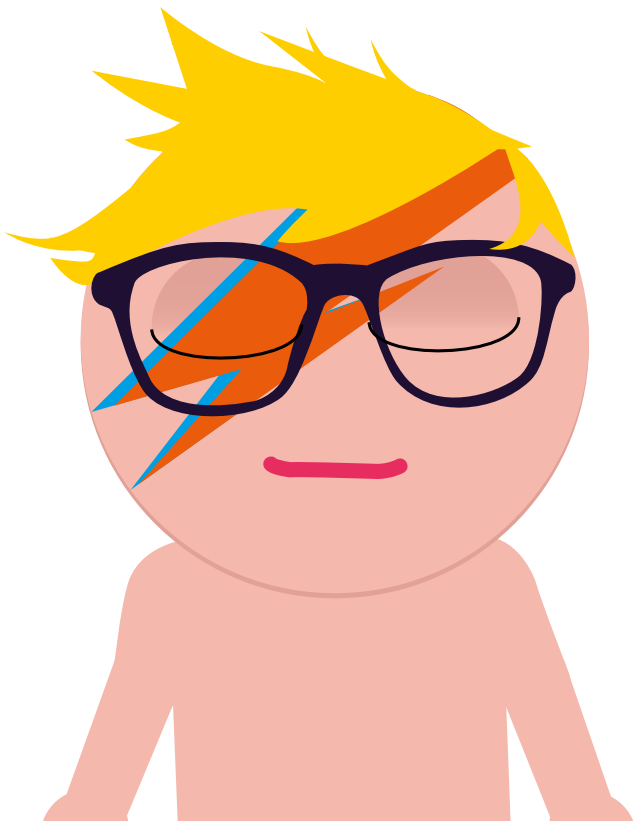
27 fairphonelink

28 indiephone.eu

Lost in Space

If you are lost in the vast space of mobile development, do not worry, stay calm and keep on reading. Go through the options and take the problem that you want to solve, your target audience and your know-how into account. Put a lot of effort into designing the experience of your service, concentrate on the problem at hand and keep it simple. It is better to do one thing well rather than doing 'everything' only so-so. Invest in the design and usability of your solution. Last but not least, finding the right niche is often better than trying to copy something that is already successful. This guide will help you make an informed decision!





From Idea To Concept

Developing popular and innovative digital solutions is a major gain in our industry. There are millions of apps in app stores, many clustered into a cloud of similar apps offering nothing special. However, some apps are outstanding. You may have asked yourself: What makes these apps successful, and how can I achieve similar success with my products? This chapter gives you tools and a framework to systematically generate innovative product ideas.

The Key Elements of Success: Desirability, Viability & Feasibility

Let's start with a discussion about what makes a product innovative and successful. A major characteristic of innovative products is their combination of three major aspects: human desirability, financial viability, and technical feasibility.

Human desirability presents a challenge to answering the questions: What do people desire, and what creates joy for them? On the one hand, this means a good idea should create value for people in their daily lives; a good idea should simplify tasks or make people's lives more comfortable. In that way, a product idea generates interests and attracts people to use the product. On the other hand, an attractive user experience fosters the joy of use, since tasks can be easily fulfilled, the structure of the application is very clear and easy to understand, and regular updates on the content or featured site continuously drum up interests in the product. Mobile apps can enable users to do things they couldn't easily do before, for instance price comparisons in store, buying and selling while on the move, monitoring their health and fitness.

Consider ways to develop apps that take advantage of what mobile devices offer and enable them to do.

The aspect of financial viability is crucial for development of most products, including mobile apps. Financial viability helps cover the costs of your time and energy in developing and maintaining your mobile app. For those hoping to make the app pay for itself, the challenge is to define a business model that enables you to create revenue from an idea and maintain acceptable costs for your customers. However, many mobile app developers remain below the poverty line¹.

Particularly, in the digital world, various new business models have emerged during the last decade. For example, Targeted Ads and Freemium business models both apply to mobile apps, with In App Purchases being particularly popular for mobile games, albeit with a potential backlash². See the monetization chapter of this guide to learn more about your options how to earn money with mobile software.

The third aspect is technical feasibility. Most software engineers and developers are involved in evaluating this dimension. Often, it is a challenge to build and combine the right technology to make a product alive. Real-life examples show that innovative products do not always need cutting-edge technology to be successful and that a smart combination of existing technologies can yield innovative products.

It is important to consider all three aspects (desirability, viability, and feasibility) to develop a fairly detailed concept before spending unnecessary efforts in implementing a solution. Get feedback from others, including potential users to help refine your idea and concept.

¹ developereconomics.com/reports/developer-economics-q3-2014/

² developereconomics.com/freemium-apps-killing-game-developers/

Put the User Into Focus

Theoretically, it is possible that a product directly meets the desires of your target group. However, in practice, such effects are rare. In most cases, the product does not please many of the intended customers. Although the features offered by the product may be very innovative, cool, and actually very useful, users may see the product as unsuitable for their context or may need various additional features to make real use of the product. In order to enhance user satisfaction, software companies then tend to make adjustments and try to implement all wishes and features of the target group in an unorganized way. Thus, the product loses its simplicity, is not very usable, and loses more and more users, since the expectations of the users cannot be met with the existing implementation.

Starting developments without really understanding user needs is highly risky because changes in an implemented app are expensive. Moreover, resources are wasted and unnecessary features build .

Define the User's Needs

In order to avoid such failures, it is important to focus on the user and develop the app using their feedback. The so-called user requirements analysis describes the crucial processes of revealing user needs and determining user expectations. Analyzing the problem space and understanding user requirements are integral parts of the design of innovative digital solutions.

The first step is to know who your users are and to define target group(s) for your app. Understand what goals the users want to achieve, what tasks they need to fulfil, and why your app is relevant to their needs. Verify that you know the groups of people who would like to use your app.

In order to reveal real pain points and to derive real

requirements, it is necessary to understand how users perform relevant tasks right now including any current workarounds. The best way to obtain necessary insights into real user needs is to speak directly to representatives of the user groups and observe them in their daily lives or work. Secondary market research, such as reports or demographic information, may augment your direct research, please do not rely on it as the primary source of information!

Furthermore, during requirements analysis, it is important to consider the differences between wishes and needs. According to Merriam Webster, a wish is "a desire for something to happen or be done."³ Additionally, it is "an act of thinking about something that you want and hoping that you will get it or that it will happen in some magical way." This desire is normally conscious. Wishes focus on concrete material objects (i.e. smart phones) or on abilities (i.e. creativity). This is why users can express their wishes for a special product, like special features or colors. Wishes can be changed (i.e. by advertising or when better products become known). If a new product reflects only users' wishes, it does not necessarily support users in fulfilling their tasks. For example, some customers want to obtain a product only because it has a nice design.

In contrast, needs reside behind wishes. They do not focus on objects or abilities but on emotional factors, like appreciation as a human, trust, or competence. According to Merriam Webster, a need is "something that a person must have" and "something that is needed in order to live or succeed or be happy."⁴ A need is a desire based on lack. If a person experiences a lack of something, this creates a desire deeply in the subconscious. This desire motivates actions, which should

³ www.merriam-webster.com/dictionary/wish

⁴ www.merriam-webster.com/dictionary/need

eliminate the lack. Different lacks induce varying degrees of actions. According to the famous Maslow's need pyramid⁵, a physiological need for sleeping or hunger is, for example, much stronger than a need for social communication. In contrast to wishes, needs are unspecific and often unconscious.

To start your requirements analysis, make sure to present needs and to define your product based on real needs instead of arbitrary wishes.

Ideating

The result of analysis is not a solution but a clear and well-founded problem statement. This problem statement should be a foundation to explore the solution space in the ideation phase of the project. During this phase, it is important to start thinking very divergently and come up with a huge amount of ideas. If you create a large amount of idea, it can be easier to discuss different approaches to solve the users' pain points and combine good ideas.

To encourage creative thinking, many creativity techniques exist. Several techniques are well known such as brainstorming. However, many other techniques have been developed and can be used in specific situation, such as the 6-3-5 method⁶, visual confrontation, or the Disney method⁷. What all of these techniques have in common is that they support divergent and convergent thinking and encourage out-of-the-box thinking.

One piece of general advice is to hold creativity sessions in a group with 5-8 participants. A group usually comes up with better results than those of an individual person. Classical

⁵ see en.wikipedia.org/wiki/Maslow's_hierarchy_of_needs

⁶ a group structured brainwriting technique, see en.wikipedia.org/wiki/6-3-5_Brainwriting

⁷ developed by Robert Dilts in 1994, see en.wikipedia.org/wiki/Disney_method

brainstorming is the most popular method, but it does not produce the best results regarding the quantity of ideas. To achieve better outcomes, we recommend the application of brainwriting techniques, which allow each participant to collect ideas alone before discussing them in a group. This encourages every team member to actively participate in brainstorming and reduces the risk that ideas are evaluated too early.

No matter which technique is used, it is important to consider the following rules for the team during the ideation phase of the project.

- Defer judgment
- Encourage wild ideas
- Stay focused
- Go for quantity
- Be visual
- Build on the ideas of others

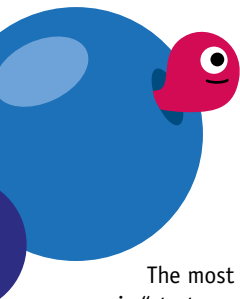
Once various ideas are collected, they should be discussed, refined, or combined in the team, and the most promising ideas should be selected.

Proving Ideas

After the selection of an idea, it is crucial to find out if the idea provides real value for the target group and if it actually meets users' needs. Feedback from your target users is crucial to evaluating the value of your idea. At first, you can simply talk about your idea and the general concept of your app. However, it can be challenging to describe your thoughts in a way in which your audience understands them as you mean it. Natural language is often interpreted differently and creates different mental models for each person. Thus, understand-

ings can be diverse, and communication about ideas can be challenging.

In order to overcome the challenges of communication, you should express your ideas in visual and tangible ways. You can prototype your ideas. The term prototype might sound like something that takes a lot of effort to build. In fact, traditional understanding of a prototype views it as a pre-version of a final product. A more modern view of prototypes is more versatile and also includes drafts (for instance, a sketch on paper) . In the early phases of developing an innovative app, a prototype should be a tool that is used to discuss the app's ideas and concepts. A prototype can be, for instance, a sketch, storyboard, or physical visualization of a concept. It should be easy and fast to build and represent the essential parts of your idea.



The most important principle for the work with prototypes is “start small, fail early, and learn fast.” This means you can create a fast prototype with very low costs to make your idea tangible, get feedback on the idea, and learn from the feedback in very short iteration cycles. In these iterations, you can test and enhance the prototype very quickly with low costs. In this way, it becomes possible to validate new ideas with low risks and without costs for the implementation of the project.

When it comes to software engineering, there are three artifacts related to the term prototype: wireframe, mock-up, and proof-of-concept implementation. Wireframes and mock-ups are usually used in early phases to validate a concept, information architecture, and basic interaction. Proof-of-concept implementations are used to validate technical feasibility. During the proof-of-concept implementation, it is crucial to focus on the risks in the project and elaborate technical possibilities and boundaries. See the following chapter to learn more about how to create prototypes.

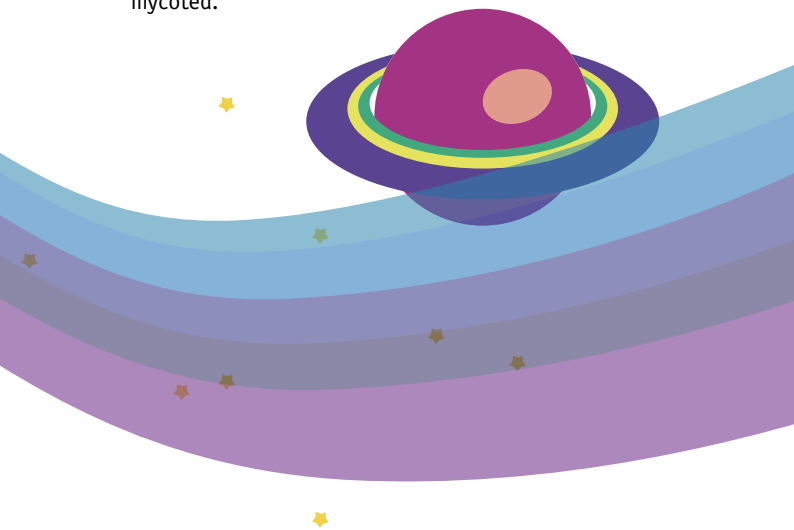
If desirability and feasibility are validated, it is also important to think about the business model for your innovative digital solution. Tools like the Osterwalder Business Model Canvas⁸ can help build your business case in a structured way.



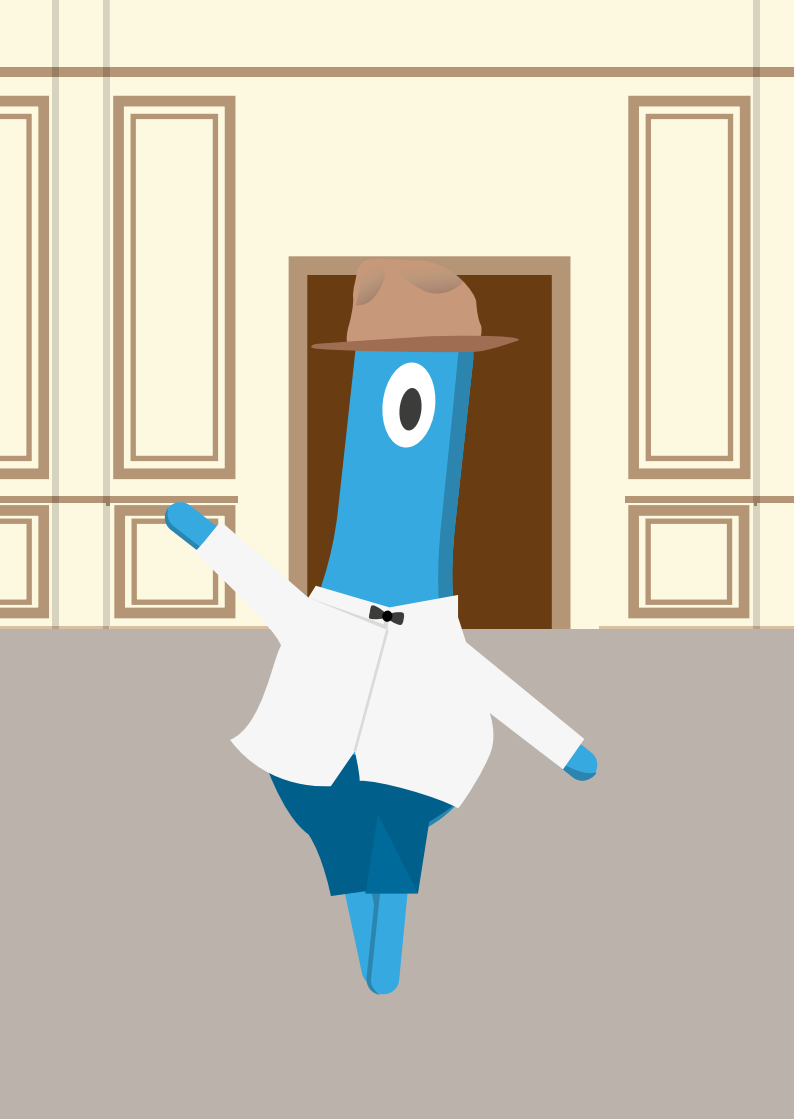
Learn More

Here is where you can learn more about the outlined methodology and techniques:

- *businessmodelgeneration.com*, the website of the famous book about how to generate your individual business model. They also offer an online tool, a free preview of the book⁹, and a lot more.
- *mycoted.com/Brainwriting*, a good introduction into Brainwriting methods. Part of a comprehensive Wiki about creative techniques maintained by the UK-based company mycoted.



⁹ businessmodelgeneration.com/book



User Experience & Interface Design

At this point you probably have an idea for your app. This chapter is about the techniques that are useful for transforming your concepts into a working and usable product.

Designing an application from scratch requires a different approach than working with an existing user interface (UI). In both situations you will frequently switch between big-picture and detailed design mindsets to make sure that you continue to improve quality while staying on track to complete the project.

You should use the methods described here as a toolbox rather than a recipe and apply them in any order that works best for a specific project that you are working on.

The design steps - conceptual thinking, applying layers of detail to the interface and verifying your concepts with users - form a cycle that you can repeat as many times as you need (or have time for).

Conceptual Thinking

Personas

Persona is a fictional character profile that represents a potential user group for a new product. A typical persona will consist of background information about a user. I.e. what they do for a living, their interests and motivations for using your product and how it will improve certain area of their life. Will it help them to be more productive at work? engage with new activities? Or will they use it to learn new skills? Usually there is more than one persona for any project. To make it easier

to relate to your user needs, choose your personas names and representative pictures.

User profiles are important to help avoid a situation where you start designing for yourself. Given that you are already working on the product, your knowledge about its detail is larger than anyone else's who will be using it for the first time. Even if your personal profile happens to fall exactly into one persona's category it is probably not the only user type that you are trying to reach. While you are going to put in the time and effort to understand and design your interface, it will almost never be the case for your users. Instead they will judge your app's UI based on how it matches their assumptions of it. To make sure that you are communicating the functionalities clearly, keep your personas in mind and ask the questions that they are likely to have on each step of their journey through your app.

User journeys

Once you get a better idea of the kind of people you are designing for, you can start mapping out functionality around specific scenarios. Thinking in flows - a specified journey from point A to B within the application - is important before you move onto more defined interface design. Outlining flows early will help you to define how many interactive steps and application screens you actually need for a specific user journey.

Examples of a journey would be an item purchase flow, uploading and sharing a photo, making a doodle sketch etc. To map them you can use flowcharts, abstract diagrams, mockups, representative screens layouts, written descriptions of each step, storyboards or simple annotated illustrations in a comic-book-style. Whichever technique you choose as the most appropriate for the current stage of design you can make your journey work not only for envisioning new products but also

evolving existing features. Flows can be detailed and use accurate designs or abstract, focused only on a specific element that you are looking at, like messaging across the app.

After outlining the main journey you can break it down to several smaller tasks that you need to consider on a particular step of the flow. This will give you the focus that you need to design a complete experience with the comfort of having the main journey covered before diving into details.

Thinking through the same design problem repeatedly will reveal a different set of challenges depending on which persona you have currently in mind. Some users will need more guidance than others. Even if it is unlikely that everyone will notice every bit of information/design you put in place, as long as it is available for those that need it and balanced so it doesn't get in the way of more independent users, it will reach a wider group.

Wireframes

Wireframes are sketched layouts of your application screens. They show where each element will be placed and how important it will be in relation to other objects on the page. This basic composition makes the understanding of how complex the content and interactions on your app's screens will be.

Wireframes can be as simple as sketches on paper, or you can use one of the many available digital tools. Sketches are ideal at the early stages for quick concepts generation. The more specialized wireframing applications come with libraries of ready-made widgets to quickly arrange on a page. The advantage of using digital tools is that you have an editable version of your mockups that can be then transformed into clickable prototypes. Although you can also use sketches to do that, once you get to more detailed problems a digital design is easier to update and maintain.

Frequently updated mockups will guide your content development and visual design. They are an excellent reference point for discussing details and next steps with everyone on the team. UI pattern libraries will be based on repeatable elements from your sketches. Your application information architecture will be structured around how you choose to organize your content and navigation around it.

Applying Details

Designing for a specific platform and multiple screen sizes

Think about your layout and widget choices in the context of the platform on which your app will run. Each platform has their own styling conventions and specific methods for handling interaction. Following the recommended practices will make your app instantly easier to interact with for users familiar with their device's patterns. For more information and links to specific online resources see the platform-related chapters of this guide.

With the ever-changing mobile devices market you should consider how your UI will look on different screen sizes and resolution densities. While it may be too early to get into too much detail before you have your concept refined, thinking about the layout scalability-to-usability ratio during the wire-framing and visual design stage (so once you have some sort of graphic representation of your layouts) can save a lot of development and testing time later. If this topic is completely new to you it is worth reading more about best practices in Responsive Web Design (RWD). Web designers have been solving this problem for a while now. Check if the platform specific guidelines provides more information around this topic.

Language

As soon as you start using meaningful labels and titles rather than placeholder text, you will start setting the way you communicate with your users. Depending on your application, language will have different roles in guiding your users through the flow. But even if the use of words in your UI is minimal, do not leave it in a placeholder state too long. Your wording is something that you should put through user testing. A single misleading word might confuse your users or lead them to assumptions that might be damaging to what you are trying to achieve.

Visual Design

Unless you are building an app that uses a non-visual input/output, your app UI will rely on graphics. Taking care of visual design details will help improve your app's experience and make it stand out among the masses.

There were a number of graphic design principles already applied during wireframing. These include spacing and visual hierarchy as well as layout structure. Polished visual design styles will not only improve your UI's visual appeal in the aesthetic sense, well executed branding enforces your app functionality and reduces the learning curve for users by providing clear visual cues.

Style consistency through the flow helps users make sense of your UI and learn interactions faster. For example, if your main action button changes color from screen to screen, consider the impact on the users. Will they be confused? Will they understand the significance of the change? If the style changes are intentional make sure you are doing them for good reasons.

Similar to designing layouts and interactions on the wireframes level, certain styling decisions might be informed by a specific platform guideline. Your app can look very different depending on which platform it was designed for. Make sure that your design follows the recommended practices for font use, standard icons and layout conventions. Again, see the platform-related chapters of this guide to find more information and links to specific resources.

Company branding in the UI can be applied in a non-obstructive way so that users can concentrate on interacting with you app. Use the background, control colors and maybe certain images or layout choices to add the desired look and feel. A splash screen (if present) is the place where you can display some additional graphics.

Finally, the launch icon is the first-impression visual element that your app will be identified by and judged on. Make it look good. If you are planning releasing on multiple platforms, check the design requirements early so you can come up with portable artwork.

Prototyping

Interactive prototype is the best way to visualize and evaluate your app's interactions. Because it is responsive enough to communicate the vision you do not need to provide as much documentation as with static images. Your prototype can have visual design in place and look exactly as it will after the implementation, or you can stay on the wireframe level.

It does not matter whether you have a big budget or are working on a personal project over the weekend, having a fairly complete prototype of your app is the best way to communicate your concept and discuss it with others. The non-linear narration of your apps should be self explanatory at this stage. Via some prototypes you can experience your concept

on an actual device. UI designs can look/feel different on large and mobile device screens.

There is no single best way of putting a prototype together. You can use whatever technique works for you. From paper prototyping to using one of the specialized tools or other applications that have the functionality to put clickable journeys together. If you have coding skills, building a HTML prototype is a good way to go. You can rapidly prototype on the existing app and work directly in final code, it all depends on what approach works for a specific project setup.

Prototypes are usually developed before you spend time on implementing code and pixel perfect designs. An agreed clickable walkthrough is a useful reference that teams can work towards without risking going too much off track. It is also great to user test prototypes and get external feedback on.

Some available tools are free and most of the commercial ones offer trial version or have free account options for limited number of projects. Here is a list of a few applications that you can try:

Prototyping & wireframing tools

- **Axure:** axure.com
- **Balsamiq Mockups:** balsamiq.com
- **Framer:** framerjs.com
- **Mockingbird:** gomockingbird.com
- **OmniGraffle:** omnigroup.com/products/omnigraffle
- **Origami:** facebook.github.io/origami
- **Pencil:** pencil.evolus.vn
- **POP:** popapp.in
- **Proto io:** proto.io
- **Sketch:** designcode.io/sketch

User Testing

The best way to validate your interface concept is to show it to users as soon as you have enough to receive feedback. You do not have to wait until you have a finished and polished product. Testing early can save you a lot of time in the long term, because it will expose concepts that don't work early in the process. The more time you invest into developing your designs, the harder it gets to let go of them and start over. It is more difficult to accept feedback on something that you considered almost done than on a clickable prototype that you can update quickly.

Typically user testing is about an hour long session. During that time users that are unfamiliar with the product are asked to perform certain tasks, usually around core functionality. When searching for people to interview it is good to refer to the original persona descriptions and look for users that match those profiles.

To make best use of the testing time, prepare in advance. Note introductions, think of how you will explain the session to users and how you will use their feedback. You should also prepare the tasks that will correspond to what you want to test. List them and have them handy to ensure that you cover them all.

If you are testing a prototype it is worth mentioning to users that the software is not complete and there might be some unfinished parts. If they assume that the person that running the session is the author of the design they might feel cautious of giving critical feedback. Reassure them that they are free to express their honest opinions. After all, the only reason you arranged testing the session is to get independent feedback. Once the expectations are set it is important that you follow the rules and not lead users to any conclusions. Do

not help them out too much (unless you cannot proceed with the session) and word your questions in a non-interrupting way. During the session either record user feedback or make sure you take enough notes.

When you get feedback, you can reiterate your designs and improve the parts that were not quite complete or move to the development with more confidence. If you are exploring new areas and the prototype is not quite ready, you can run testing sessions on other apps that are currently released. It can surprise you how much others notice about the application that you might have never thought of.

Even if you are unable to test with a large number of people, testing with only several users will raise the major issues that are the most likely to cause usability problems. Do not be afraid to undertake these sessions whenever it makes sense to validate the direction you are heading in. A single non-biased opinion is better than no opinion at all.

Learn more

There is plenty of resources available online. Here are some to whet your appetite:

- **UX Archive:** uxarchive.com
- **User Onboarding:** useronboard.com
- **Smashing Magazine** (UX design section):
uxdesign.smashingmagazine.com
- **UX Magazine:** uxmag.com
- **UX Matters:** uxmatters.com
- **Nielsen Norman Group:** nngroup.com
- **Interaction Design Foundation:** interaction-design.org



Android

The Ecosystem

The Android platform is developed by the Open Handset Alliance led by Google and has been publicly available since November 2007. Its use by the majority of hardware manufacturers has made it the fastest growing smartphone operating system. More than 84% of all smartphones sold in Q3 2014 worldwide were based on Android¹. At the 2014 event of the annual Google I/O, Google announced that over 1 billion Android devices have been activated so far² which also includes wearables, tablets, media players, set-top boxes, desktop phones and car entertainment systems. Google's own smart eyeglasses, Google Glass, runs a minimal version of Android supporting both web and native apps. Some non-Android devices are also able to run Android applications with reduced functionality, such as RIM's Playbook with its BlackBerry Android runtime, the new Open Source OS Sailfish³ and the crowdfunded gaming console Ouya.

In December 2014, the number of Android apps on Google Play cracked the 1.5 million barrier⁴.

Android is an operating system, a collection of pre-installed applications and an application framework supported by a comprehensive set of tools. The platform continues to evolve rapidly, with the regular addition of new features every 6

¹ idc.com/prodserv/smartphone-os-market-share.jsp

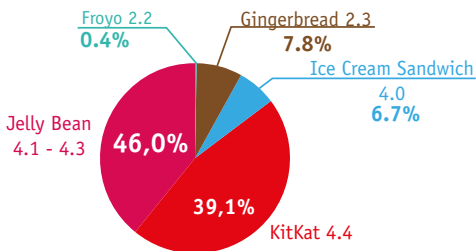
² engadget.com/2014/06/25/google-io-2014-by-the-numbers

³ sailfishos.org

⁴ www.appbrain.com/stats/number-of-android-apps

months or so with the newest release being Android 5.0 (codename 'Lollipop'). Being the latest major release after a handful of minor releases, Lollipop was announced as the largest and most ambitious release yet. A new UI toolkit called Material Design⁵ aims to unify all targeted platforms like phones, wearables and TVs with one design approach. The former runtime Dalvik is replaced with the new Android Runtime (ART) that provides features like a smarter garbage collection that ultimately improves performance by a factor of up to 4x. Project Volta introduces tools to improve the battery life by optimizing the behaviour of an app. Further features include privacy features, cross-device user accounts and extensions to the notification system.

One of the most discussed issues when developing for Android is the system's fragmentation: The multitude of different devices by various manufacturers and the fast progress of the platform itself leads to uncertainty over whether or not your Android application will run everywhere. In addition, the adaption of the latest OS version is slower compared to other mobile platforms. However, today, you will reach over 90% of the installation base if you decide to target Android 4.0 or above⁶.



(data from January 2015)

⁵ developer.android.com/design/material/index.html

⁶ developer.android.com/about/dashboards

To encourage a solid user experience and consistent appearance of Android apps, Google publishes a design guide⁷. Going into the importance of color schemes, design patterns and the new Material design, the guide provides a great orientation when building apps for the Android ecosystem.

Prerequisites

The main programming language for Android is based on Java. But beware, only a subset of the Java libraries and packages are supported and there are many platform specific APIs that will not work with Android. You can find answers to your "What and Why" questions online in Android's Dev Guide⁸ and your "How" questions in the reference documentation⁹. Furthermore, Google introduced a section in their documentation called "Android Training"¹⁰ that helps new developers learn about various best practices. This is where you can learn about basics such as navigation and inter-app communication, as well as more advanced features such as intelligent Bitmap downloads and optimizing your app for better battery life.

To get started, you need the Android SDK¹¹, which is available for Windows, Mac OS X, and Linux. It contains the tools needed to build, test, debug and analyze apps. The Android Development Tools (ADT)¹² are responsible for the integration with IDEs and making sure that your development flow is as comfortable as possible.

⁷ developer.android.com/design

⁸ developer.android.com/guide

⁹ developer.android.com/reference

¹⁰ developer.android.com/training/index.html

¹¹ developer.android.com/sdk

¹² developer.android.com/tools/sdk/eclipse-adt.html

IDE support

Today, Google offers prepacked IDEs based on IntelliJ called "Android Studio", and Eclipse (referred to as "Eclipse + ADT Plugin"), effectively bundling the Android Developer Tools with the IDE. After 2 years of development, Android Studio 1.0¹³ is now the official IDE for Android and comes directly with Gradle-support and many features directly tailored to Android development.

IDE	plugin support	bundled version
Eclipse	seperate ADT package	Eclipse + ADT Plugin
IntelliJ	seperate Android plugin	Android Studio

More information and the required downloads can be found in the Android documentation's "Tools"¹⁴ section.

Native development

The Android NDK¹⁵ enables native components to be written for your apps by leveraging both JNI for invocations of native methods and using native subclasses that offer callbacks to it's non-native pendants. This is important for game developers and anyone who needs to rely on efficient processing.

Implementation

App Architecture

Android apps usually include a mix of Activities, Services, BroadcastReceivers and data providers; these all need to be declared in the application's manifest.

¹³ android-developers.blogspot.de/2014/12/android-studio-10.html

¹⁴ developer.android.com/tools

¹⁵ developer.android.com/tools/sdk/ndk

An Activity is a piece of functionality with an attached user interface. A Service is used for tasks that run in the background and, therefore, are not tied directly to a visual representation. A Message Receiver handles messages broadcast by the system, your own or other apps. A Data Provider is an interface to the content of an application that abstracts from the underlying storage mechanisms (e.g. SQLite).

An application may consist of several of these components, for instance an Activity for the UI and a Service for long running tasks. Communication between the components is achieved by Intents or remote procedure calls handled by Android Interface Definition Language (AIDL).

Intents bundle data, such as the user's location or a URL, with an action. These intents trigger behaviors in the platform and can be used as a messaging system in your app. For instance, the Intent of showing a web page will open the browser. A powerful aspect of this building block philosophy is that any functionality can be replaced by another application, as the Android system always uses the preferred application for a specific Intent. For example, the Intent of sharing a web page triggered by a news reader app can open an email client or a text messaging app depending on the apps installed and the user's preference: Any app that declares the sharing Intent as their interface may be used.

The user interface of an app is separated from the code in Android-specific XML layout files. Different layouts can be created for different screen sizes, country locales and device features without touching the Java code. To this end, localized strings and images are organized in separate resource folders. Of course, you are also able to define and design layouts in code or make use of both strategies to enable dynamic UI updates.

The SDK and Plug-Ins

To aid development, you have many tools at your disposal in the SDK, the most important ones are:

- **android:** To create a project or manage virtual devices and versions of the SDK.
- **adb:** To query devices, connect and interact with them (and virtual devices) by moving files, installing apps and alike.
- **emulator:** To emulate the defined features of a virtual device. It takes a while to start, so do it once and not for every build.
- **ddms:** To look inside your device or emulator, watch log messages, and control emulator features such as network latency and GPS position. It can also be used to view memory consumption and kill processes. If this tool is running, you can also connect the Eclipse debugger to a process running in the emulator. Beyond that, ddms is the only way (without root-access) to create screenshots in Android versions below 4.0.

These four tools along with many others, including tools to analyze method trace logs, inspect layouts and test apps with random events, can be found in the tools directory of the SDK.

IDE plug-ins are available to help manage all these files. Version 11.x of IntelliJ includes a visual layout-editor, so you are free to choose between Eclipse and IntelliJ should you want to perform rapid prototyping by dragging UI-elements in the editor.

If you are facing issues, such as exceptions being thrown, be sure to check the ddms log or use the logcat mechanism. It enables you to check whether you neglected to add all necessary permissions, for example,

`android.permission.INTERNET` in the `uses-permission` element¹⁶.

If you are using features introduced after Android 2.3 such as Fragments¹⁷ for large screens, be sure to add the Android Compatibility package from Google. It is available through the SDK and AVD Manager and helps development for Android 3.0+ without causing problems with deployment to Android 1.6¹⁸ through to Android 2.3. Be sure to use the v4 packages in your apps to provide maximum backwards support. There is also a version for Android 2.1 and above called v7 appcompat library that introduces a way to implement the ActionBar pattern and more as documented online¹⁹.

Developing your application against Android 3.1+, will enable you to make homescreen widgets resizable, and connect via USB to other devices, such as digital cameras, gamepads and many others. Android 4.X releases introduced further interesting features such as expandable notifications, lockscreen widgets, and a camera with face detection. The Material Design UI Toolkit was introduced with Android 5.0 and introduces new widgets and more to use in phones, wearables and other platforms. The native computing framework, Renderscript (introduced in 3.1), was heavily changed and no longer provides direct graphic rendering capabilities but may now be used for heavy processing instead.

To provide some backwards compatibility for devices with older Android versions, Google began to use the Google Play Services framework²⁰ which gets updated via the Play Store

¹⁶ developer.android.com/reference/android/Manifest.permission.html

¹⁷ developer.android.com/guide/topics/fundamentals/fragments.html

¹⁸ android-developers.blogspot.com/2011/03/fragments-for-all.html

¹⁹ developer.android.com/tools/support-library/features.html

²⁰ developer.android.com/google/play-services/

and adds libraries such as the latest Google Maps. If you are interested in authenticating users, you might want to have a look at the Google+ Sign capabilities that bring the benefit of real user data to your app. The functionality is managed via OAuth 2.0 tokens that allow use of the Google Account on the user's behalf.

Testing

The first step in testing an app is to run it on the emulator or a device. You can then debug it, if necessary, through the ddms tool.

All versions of the Android OS are built to run on devices without modification, however some hardware manufacturers may have changed pieces of the platform. Therefore, testing on a mix of devices is essential. To get an idea of which devices are most popular, refer to AppBrain's list²¹.

To automate testing, the Android SDK comes with some capable and useful testing instrumentation²² tools. Tests can be written using the standard JUnit format, using the Android mock objects that are contained in the SDK.

The Instrumentation classes can monitor the UI and send system events such as key presses. Your tests can then check the status of your app after these events have occurred. MonkeyRunner²³ is a powerful and extensible test automation tool for testing the entire app. These tests can be run on both virtual and physical devices.

In revision 21 of the SDK, Google finally introduced a more

²¹ www.appbrain.com/stats/top-android-phones

²² developer.android.com/guide/topics/testing/testing_android.html

²³ developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

efficient UI automation testing framework²⁴ which allows functional UI testing on Android Jelly Bean and above. The tool itself can be executed from your shell with the command `uiautomatorviewer` and will present you the captured interface including some information about the views presented. Executing the tests is relatively easy: After you have written your test, it is then built via ANT as a JAR-file. This file has to be pushed onto your device and then executed via the command `adb shell uiautomator runtest`.

In October 2013 a new tool called Espresso²⁵ was released by Google. It provides a very lean API that helps to quickly write procedural tests for your UI.

Open source testing frameworks, such as Robotium²⁶, can complement your other automated tests. Robotium can even be used to test binary apk files if the app's source is not available. Roboelectric²⁷ is another great tool which runs the tests directly in your IDE in your standard/desktop JVM.

Your automated tests can be run on continuous integration servers such as Jenkins or Hudson. Roboelectric runs in a standard JVM and does not need an Android run-time environment. Most other automated testing frameworks, including Robotium, are based on Android's Instrumentation framework, and will need to run in the Dalvik JVM. Plugins such as the Android Emulator Plugin²⁸ enable these tests to be configured and run in Hudson and Jenkins.

²⁴ android-developers.blogspot.de/2012/11/android-sdk-tools-revision-21.html

²⁵ googletesting.blogspot.de/2013/10/espresso-for-android-is-here.html

²⁶ code.google.com/p/robotium

²⁷ robolectric.org/

²⁸ wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin

Building

Aside from building your app directly in the IDE of your choice, there are also more comfortable ways to build Android apps. Gradle²⁹ is now the officially supported build automation tool for Android. There is also a maven plugin³⁰ which is well supported by the community. Both tools can use dependencies from different Maven repositories, for example the Maven Central Repository³¹.

Google ships libraries for Gradle as Android Archive (.aar) files that can be obtained using the Android SDK Manager. You are also able to package your own libraries or SDKs utilizing the android-library plugin for Gradle. A great source for finding Gradle-friendly Android libraries is "Gradle, please"³².

Signing

Your apps are always signed by the build process, either with a debug or release signature. You can use a self-signing mechanism, which avoids signing fees (and security).

The same signature must be used for updates to your app - so make sure to not lose the keystore file or the password. Remember: you can use the same key for all your apps or create a new one for every app.

²⁹ tools.android.com/tech-docs/new-build-system

³⁰ code.google.com/p/maven-android-plugin/

³¹ www.maven.org

³² gradleplease.appspot.com

Distribution

After you have created the next killer application and tested it, you should place it in Android's appstore called "Play". This is a good place to reach customers and sell them your apps. Android 1.6 upwards also supports in-app purchase through Google Wallet. This enables you to sell extra content, feature sets and alike from within your app by using the Android Play³³ infrastructure. It is also used by other app portals as a source for app metadata. To upload your application to Android Play, go to play.google.com/apps/publish/.

You are required to register with the service using your Google Checkout Account and pay a \$25 registration fee. Once your registration is approved, you can upload your app, add screenshots and descriptions, then publish it.

Make sure that you have defined a `versionName`, `versionCode`, an icon and a label in your `AndroidManifest.xml`. Furthermore, the declared features in the manifest (uses-feature nodes) are used to filter apps for different devices.

One of the recent additions to the Google Play Store is alpha and beta testing plus staged rollouts. This allows you to do some friendly user testing before publishing the app to all users. Furthermore, you can target specific countries and devices by setting the right flags in the Developer Console and export detailed statistics that help in understanding your userbase. Using the inbuilt localization service, you can easily add new languages to your app by paying a fee - make sure to check the Localization Checklist³⁴ for detailed information about the importance of this topic.

³³ developer.android.com/guide/google/play/billing/

³⁴ developer.android.com/distribute/googleplay/publish/localizing.html

As there are lots of competing applications in Android Play, you might want to use alternative application stores³⁵. They provide different payment methods and may target specific consumer groups. One of those markets is the Amazon Appstore, which comes pre-installed on the Kindle Fire tablet family.

Adaptation

As adaptation of Android increases, vendor specific ecosystem have also been growing that involves their own SDKs, fully-customized Android versions and tools around topics such as alpha and beta testing. This has both upsides, such as a very tight integration that allows an amazing experience for users, and downsides, such as increased fragmentation of ecosystem. Vendor specific marketplaces often prohibit the upload of generic apps that utilize utilities other than their own.

One example is Amazon's Kindle Fire ecosystem which is basically a customized fork of Android and represents the Android tablet with the biggest market share: Instead of using Google's Play Services for enabling in-app purchases or maps, you have to use Amazon's own libraries that offer similar functionality. The reasoning behind it is pretty simple: Kindle devices are not delivered with the required libraries to run Google's services. Amazon also offers their own advertisement and gaming services (comparable to Google Play Games) that help to target your audience. Offering Emulators for their four different devices (1st Gen, 2nd Gen, HD 7" and HD 8.9"), Amazon helps perfect your app by providing a realistic environment. On top

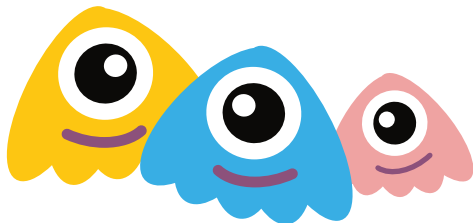
³⁵ onepf.org/appstores/

of the testing that Amazon offers their developer community, they also review any apps that get uploaded to their Appstore.

Here is a little overview that can help you find the right resources:

Vendor	Documentation
Amazon	developer.amazon.com/sdk/fire.html
HTC	htcdev.com
LG	developer.lge.com
Motorola	developer.motorolasolutions.com/community/android
Samsung	developer.samsung.com/android
Sony	developer.sonymobile.com

Interestingly enough more and more vendors (e.g. Samsung and HTC) have also started to offer vanilla Android versions of their devices called "Google Play Edition". These devices use the same hardware as the regular models but do not come with any software customization. These devices are directly distributed through Google's Play Store and offer bleeding edge devices to users that want to stick to Google's experience.



Monetization

In addition to selling an app in one of the many app stores available, there are several different ways of monetizing an Android app. One suitable way is by using advertising, which may either be click- or view-based and can provide a steady income. Other than that, there are different In-App Billing possibilities such as Google's own service³⁶ that utilizes the Google Play Store or PayPal's Mobile SDK³⁷ and Mobile Payments Library³⁸. Most services differ in transaction-based fees and the possibilities they offer for example subscriptions, parallel payments or pre-approved payments. If you're looking to bring extra cool functionality to your app, you should consider implementing card.io's SDK³⁹ for camera-enabled credit card scanning.

For the vendor specific ecosystems, such as Samsung Apps or Amazon's Appstore, you should consider using their SDKs to enjoy the benefits of optimized monetization.

Be sure to check that the payment method of your choice is in harmony with the terms and conditions of the different markets you want to publish your app to. Those particularly for digital downloads, for which different rules exist, are worth checking out.

³⁶ developer.android.com/google/play/billing/

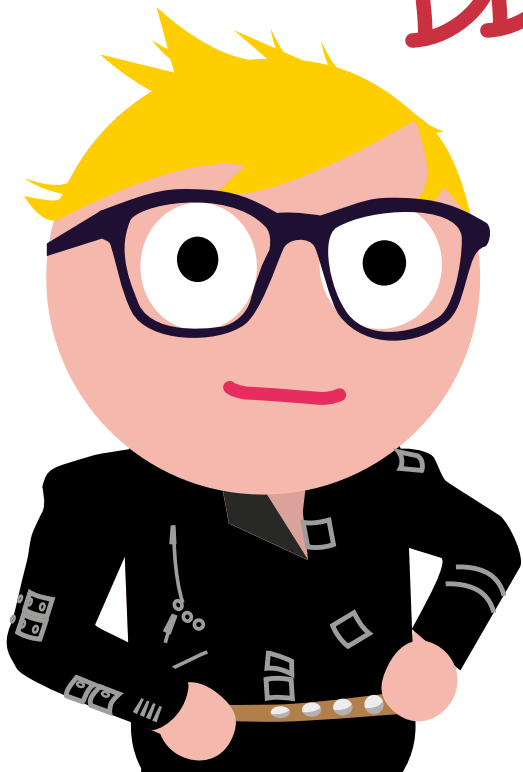
³⁷ github.com/paypal/PayPal-Android-SDK

³⁸ developer.paypal.com/webapps/developer/docs/classic/mobi

³⁹ www.card.io



BB10



BlackBerry 10

Introduction

The BlackBerry 10 platform (BB10) is a general relaunch from BlackBerry (company former named RIM). BlackBerry has taken this approach in order to catch-up with competing mobile operating systems: iOS, Android and Windows Phone 8. BB10 devices came to market in Q1 2013 - there are no upgrade plans for older generation devices. Currently nine BB10 handsets are available: The new flagships are now the Blackberry Passport and the Blackberry Classic. Blackberry provides handsets with physical or software keyboards.

Although the OS is new, the core it is based on QNX, a realtime OS for embedded devices. The other parts of the BlackBerry ecosystem, like the Marketplace called BlackBerry World or the push-service, have not changed. They have partnered with Amazon for a broader app choice. With the release of Version 10.3 of the Blackberry OS, users have the opportunity to choose between the Blackberry World Appstore and the Amazon Appstore. A big asset for BlackBerry in enterprises is the Mobile-Management-Software called BlackBerry Enterprise Server.

The latest BlackBerry SDK version 10.3.1 was released in November 2014.

Development

With BB10, apps can be developed using a wide variety of software technologies:

- C Native SDK
- C++ Cascades SDK
- HTML5 (WebWorks SDK)
- Adobe Air
- Android Runtime
- BlackBerry App Generator

To attract developers to their new OS, RIM provides a rich set of resources including a simulator and frequently updated documentation¹. There is also a free eBook Learn BlackBerry 10 App Development². And in terms of code, lots of examples Online³ and various sample projects on GitHub⁴.

A major point of discontent, for which RIM has received a lot of backlash, is that the previous Java API is not supported on BB10. This means that Java developers writing code for older, non-BB10 BlackBerry devices need to re-orient themselves to one of the technologies previously mentioned. As not all developers are willing to do this, there is concern in the community that too many developers will "jump ship" and re-orient themselves to competing platforms. Furthermore, since there is no migration path for current generation apps, developers will need to rewrite them for the new platform. This

¹ developer.blackberry.com/platforms/bb10

² apress.com/9781430261575

³ developer.blackberry.com/develop/platform_choice/bb10.html

⁴ github.com/blackberry

is necessary because the core of the new OS is based on QNX⁵, a realtime embedded OS. On the other hand, the new platform offers new opportunities, e.g. for web developers and Android developers who can easily migrate their apps. However, some migrated apps have little value and may simply be bulking out the app store rather than developing apps users want⁶.

C Native SDK

The BlackBerry native SDK supports open standards that allow developers to bring their existing apps to the platform. To get started, there is a Native Dev Site⁷. Writing your code with the native SDK enables your app to run closer to the hardware and thereby improve the performance and integration with BB10. The BlackBerry 10 native SDK includes everything you need to develop programs that run under the BlackBerry 10 OS: a compiler, linker, libraries, and an extensive Integrated Development Environment (IDE). It is available for Windows, Mac and Linux.

The core development steps are the following:

- Request a signing account and keys
- Set up the native SDK⁸
- Install and configure the simulator⁹
- Configure your environment for development and deployment
- Create your first project and/or run sample applications

⁵ www.qnx.com

⁶ mashable.com/2013/08/21/blackberry-10-app-spam/

⁷ developer.blackberry.com/native/beta

⁸ developer.blackberry.com/native/download

⁹ developer.blackberry.com/native/download

C++ Cascades SDK

Developing with C++ and Cascades is another option. Cascades has been designed to allow developers to build a BlackBerry native application with easy implementation of the GUI. The Cascades framework separates application logic from the UI rendering engine. In an application, the declared UI controls, their properties and their behavior are defined in an Markup-Language called QML¹⁰. When your application runs, the UI rendering engine displays your UI controls and applies any transitions and effects that are specified. The Cascades SDK provides the following features:

- Cascades UI and platform APIs
- Tools to develop your UI in C++, Qt Modeling Language (QML), or both
- Ability to take advantage of core UI controls and to create new controls
- Communication over mobile and Wi-Fi networks
- Recording and playback of media files
- Storage and retrieval of data
- Certificate managing and cryptographic tools

The Cascades framework is built using the Qt application framework. This architecture allows Cascades to leverage the Qt object model, event model, and threading model. The slots and signals mechanism in Qt allows for powerful and flexible inter-object communication. The Cascades framework incorporates features of fundamental Qt classes (such as QtCore, QtNetwork, QtXml, and QSql, and others) and builds on them. This lets developers define things instead of programming them e.g. they only need to define the duration and type of an anima-

¹⁰ en.wikipedia.org/wiki/QML

tion, instead of programming it. This approach is similar to iOS with Core Animation. QML can even be written by experienced Javascript developers because of its JSON-like markup.

To help developers with this new approach of UI building, there is a tool called Cascades Builder. It is built into the QNX Momentics IDE and lets developers design a UI using a visual interface. When a change to the code is made, you can see the effects immediately in the design view. The developer has no need to program a control, he can simply use a drag and drop approach.

If you are a designer, the Cascades Exporter¹¹ is for you. This Adobe Photoshop Plugin slices and rescales your images and packages them up to a tmz-File (compressed, sliced and metadata enhanced image assets). These asset files can be easily used by a developer with the QNX Momentics IDE.

To get further information, investigate the Cascades Dev Site¹².

HTML5 WebWorks

If you are a Web/JavaScript developer, you can use your existing skills to write apps for BlackBerry. There are two important tools that you can use:

The first tool is the WebWorks SDK¹³. Among other features, it allows you to write regular webpages and then package them as native BlackBerry apps with ease. The new version 2.2 of webworks fits tightly in Apache Cordova framework a.k.a Phonegap. BlackBerry published all webworks-apis as plugins for the cross-plattform Cordova tooling. If you want to mimic

¹¹ developer.blackberry.com/cascades/documentation/design/cascades_exporter

¹² developer.blackberry.com/cascades

¹³ developer.blackberry.com/html5/download/sdk

the BlackBerry-UI style in HTML, there is a project on GitHub to help you. It is called BBUi.js¹⁴ and provides extensive CSS to make your regular webpage look like a native BlackBerry-UI application. You use data-attributes to enhance the HTML for that approach. As an alternative for bbui.js there is also support for jQueryMobile with a BB10 Theme. The SenchaTouch framework¹⁵ also supports BB10.

It is good to know that RIM offers hardware accelerated WebGL support and you could do debugging and profiling on the mobile device via WebInspector as a built in feature.

To get more information about developing with WebWorks there is a HTML5 Dev micro-site¹⁶ with more information.

Adobe Air

Support for Adobe Air was removed in release 10.3.1 of BB10.¹⁷

Android Runtime

You can use the BlackBerry Runtime for Android apps to run Android Jelly Bean 4.2.2 platform applications on BlackBerry 10.2 and 10.3. To use the runtime, you must first repackage your Android applications in the BAR file format, which is the file format required for an application to run on BlackBerry 10.

As a developer, you will need to use one of the following tools to repackage your application. These tools also check how compatible your application is for running on BlackBerry 10, as some of the APIs from the Android SDK may not be supported, or may be only partially supported on the BlackBerry platform.

¹⁴ github.com/blackberry/bbUI.js

¹⁵ www.sencha.com/products/touch

¹⁶ developer.blackberry.com/html5

¹⁷ developer.blackberry.com/air/downloads/endofsupport/

- **Plug-in repackaging tool for Eclipse** The main advantage of using this tool is the ability to check for compatibility, repackage, debug, and run apps on the BlackBerry PlayBook, BlackBerry Tablet Simulator, BlackBerry 10 Dev Alpha Simulator and BlackBerry 10 device, all without leaving Eclipse. You can also use this plug-in to sign your application before it is distributed. If you want to test your application without signing it, you can use the plug-in to create and install a debug token on the target device or simulator.
- **Online packager** The main advantage of the BlackBerry Packager for Android apps is that you can use it to quickly repackage your Android application using only your browser. You can test the application for compatibility, repackage it as a BlackBerry Tablet OS or BlackBerry 10 compatible BAR file, and then sign it so that it can be distributed through the BlackBerry App World storefront.
- **Command-line repackaging tools** One of the main advantages of using the BlackBerry SDK for Android apps is that you can use it to repackage multiple Android applications from the APK file format to the BAR file format. In addition, you can also use this set of command-line tools to check the compatibility of your Android applications, sign applications, create debug tokens, and create a developer certificate.

If you want to find out more about running Android apps on BB10, please visit the dedicated website¹⁸.

¹⁸ developer.blackberry.com/android

BlackBerry App Generator

If you are not a developer, BlackBerry provides an easy way of generating a simple app for BB10 with the BlackBerry App Generator¹⁹. This webpage generates an app based on input sources like RSS feeds, Tumbler, Facebook, YouTube, flickr and more. It generates a master-detail styled app that can be customized with a logo and color selection. For a simple news-app that approach is totally fine, but do not expect any "CNN"-like masterpieces.

Testing

BlackBerry continues to provide a simulator for BB10 handsets as a separate download²⁰. This simulator enables you to run an app on a PC/Mac/Linux in the same way it would be run on a real BlackBerry device. To assist with testing, the simulator comes with a little application called controller. This utility enables you to simulate things like setting the battery level, GPS-position, NFC or tilting the device and thereby check how your application reacts in real-world scenarios.

Signing

Many security-critical classes and features of the platform (such as networking or file APIs) require an application to be signed so that the publisher can be identified. This final step in developing an app for BlackBerry is often painful.

If you like to test your unsigned app on a physical device, you need to request a file called debug token. This token enables a specific BB10 device to run unsigned apps. For this

¹⁹ blackberryappgenerator.com/blackberry/

²⁰ developer.blackberry.com/devzone/develop/simulator/

setup procedure you need to request a signing file (client-PBDT-xxxxx.csj) via the BlackBerry Key Order Form²¹. After receiving the file by email you can install a debug token with the command-line tools. After this setup you can run unsigned apps on your device. Please be advised that this needs to be done on each device separately.

If you want to publish your app in BlackBerry's AppWorld, you need a signing key also ordered through the BlackBerry Key Order Form²². To help you with this process of setup BlackBerry provides a step by step webpage²³ that guides you through the process.

Distribution

As with all previous OS versions, BB10 apps are distributed via BlackBerry AppWorld²⁴. The necessary vendor account can be created at the Vendor Portal for BlackBerry World²⁵.

For paid applications, developers get a 70% revenue share.

The second option is an enterprise distribution. This let you roll out an internal app in your organization instead of making it publicly available to any user. This is suitable for B2B Apps. If you want to find out more about enterprise distribution, please visit the dedicated website²⁶.

²¹ www.blackberry.com/SignedKeys/codesigning.html

²² www.blackberry.com/SignedKeys/codesigning.html

²³ developer.blackberry.com/CodeSigningHelp/codesignhelp.html

²⁴ appworld.blackberry.com

²⁵ appworld.blackberry.com/isvportal

²⁶ developer.blackberry.com/distribute/enterprise_application_distribution.html

FIREFOX OS



Firefox OS

The Ecosystem

Do we need another mobile operating system? Mozilla Foundation thought so and developed Firefox OS¹, a Linux based open source mobile operating system aimed at lower end smartphones. A first release has been published in February 2013.

Six months later, the first Firefox OS device for the mass market was launched: the ZTE Open offered for 80USD and mainly promoted for emerging markets. With the release of Alcatel's One Touch Fire device in Germany, Firefox OS officially entered the European market in October 2013. The phone's introductory price was set to 90 Euro. In December 2014, Mozilla and KDDI launched the first Firefox 2.0 based device in Japan² which is also promoted as the the first high-spec Firefox OS smartphone.

Firefox apps are HTML-based, but instead of packaging HTML5 web apps with tools such as Phonegap, FirefoxOS uses HTML/JavaScript/CSS as the native development languages. This means it is pretty easy for a web developer to start writing native apps for the system. You need to extend your knowledge to the JavaScript API provided by Firefox OS, and to how apps are packaged.

¹ mozilla.org/firefox/os

² blog.mozilla.org/press/2014/12/mozilla-and-kddi-launch-first-firefox-os-smartphone-in-japan-4

Firefox OS basically consists of three main components:

- **Gonk:** The low-level Linux Kernel and hardware abstraction layer (HAL). In theory a hardware vendor just needs to port the Gonk to their hardware to make it Firefox OS compatible.
- **Gecko:** The application runtime. Gecko is parses, executes and render the HTML, JavaScript and CSS. All access to the hardware needed to deliver app functionality is handled by this runtime. It includes a networking stack, graphics stack, layout engine, virtual machine (for JavaScript), and porting layers.
- **Gaia³:** The user interface (UI), written in HTML, CSS and JavaScript. Gaia provides all UI elements needed for standard dialogues. It interfaces with the operating system through Open Web APIs.

Development

There are two ways to create an app for Firefox OS: hosted apps and packaged apps. In both cases you write code in HTML, CSS and JavaScript. Hosted apps are basically a website. They are easily updated but offer limited access to the web API and need an established internet connection. Packaged apps run locally and are essentially a zip file containing all the app's assets.

Unlike normal webapps, Firefox OS apps need a manifest⁴.

³ github.com/mozilla-b2g/gaia github.com/mozilla-b2g/gaia github.com/mozilla-b2g/gaia

⁴ developer.mozilla.org/en-US/docs/Web/Apps/Manifest

This is metadata for your app which defines the name, description, icons and other information.

This is how a minimal manifest would look:

```
{
  "name": "Hello World",
  "description": "Yet another...",
  "launch_path": "/index.html",
  "icons": {
    "128": "icon.png"
  },
  "developer": {
    "name": "Your name",
    "url": "http://..."
  },
  "default_locale": "en"
}
```

The Firefox WebAPI⁵ offers you access to: vibration, geolocation, battery status, alarm, IndexedDB, proximity sensor, ambient light sensor and an archive. Using the APIs you can, for example, access the Battery Status simply by calling `navigator.battery.level` in JavaScript.

If you need more features than the WebAPI provide, you can use Activities. Mozilla uses the Object `MozActivity`, similar to Android Intents: The user will be asked which app he wants to use for a certain task.

Here is an example of how to create a short message:

```
var sms = new MozActivity({
  name: "new",
  data: {
    type: "websms/sms",
    number: "+46777888999"
  }
});
```

⁵ wiki.mozilla.org/WebAPI

While this example shows how to access the picture-gallery (picker):

```
var pick = new MozActivity({  
  name: "pick",  
  data: {  
    type:  
  }  
});
```

Simulator and Testing

Mozilla provides a downloadable simulator for Firefox OS as a browser plug-in⁶. Firefox OS 1.2 introduced the App Manager⁷. This new developer tool enables remote debugging of code and provides more GUI helpers, such as the manifest editor. However it is strongly recommended that you don't blindly trust the simulator: for example, it has far more RAM available compared to the actual Firefox devices.

Distribution

Mozilla has created an global AppStore called Marketplace⁸. Your app will be reviewed according to Mozilla's guidelines⁹. Once it is published, you will get 70% of the generated revenue.

⁶ addons.mozilla.org/en-us/firefox/addon/firefox-os-simulator/

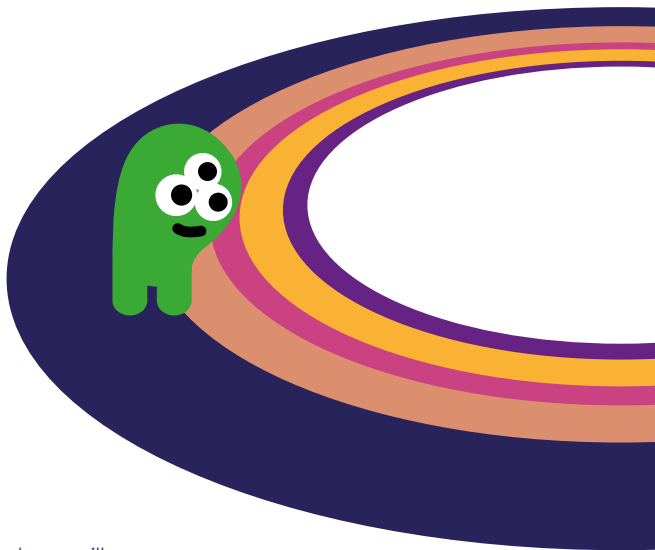
⁷ developer.mozilla.org/en-US/Firefox_OS/Using_the_App_Manager

⁸ marketplace.firefox.com

⁹ developer.mozilla.org/en-US/docs/Web/Apps/Publishing/Marketplace_review_criteria

Learn More

Your first resource to learn more about how to develop your Firefox OS app is the Mozilla Developer network¹⁰. A quick introduction, including video tutorials on how to get started, can also be found at marketplace.firefox.com/developers/docs/quick_start. André Fiedler also offers some useful insights for beginners in his presentation "Doing mobile web Apps for Firefox OS - the right way"¹¹. Finally, Mozilla's developer evangelist Chris Heilmann offers a lot of information on his blog¹².



¹⁰ developer.mozilla.org

¹¹ slideshare.net/andrefiedler1/doing-mobile-web-apps-for-firefox-os-the-right-way

¹² hacks.mozilla.org/author/cheilmann





The Ecosystem

Developing for iOS is more popular than ever. In the summer of 2014 Tim Cook, Apple's CEO, stated that there are currently 9 million registered iOS developers, a 47% increase from the previous year.¹ In January 2015, Apple announced that 1.4 million apps are available on iOS and that app billings rose another 50 percent within one year: Total app revenues on iOS in 2014 alone sum up to \$10 billion, which means that iOS developers have earned a cumulative \$25 billion from Apple's AppStore².

A popular question is where to begin on the journey to being an iOS developer. This chapter will show what is required to set up a development environment and some of the different options available for writing apps. Suggestions are given on resources to help develop your skills in the fastest time possible. The chapter ends with a few words of advice worth considering in order to become a member of the iOS Developer Ecosystem.

Birth of a Mobile OS

On January 9th 2007 Steve Jobs unveiled a new product category for the then computer and music device company. Six years later the New York times reported what was not known

- 1 techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date
- 2 apple.com/pr/library/2015/01/08App-Store-Rings-in-2015-with-New-Records.html

to the public during the announcement. As late as the day before the demo Jobs could not get through his presentation without the iPhone (only one of about 100 prototypes that existed) "randomly dropping calls, losing its Internet connection, freezing or simply shutting down"³. But when the time came for the keynote Jobs delivered it without a hitch and the rest is history.

With the launch of the original iPhone Apple unveiled a new operating system to run the device. Its original name was iPhone OS since the iPhone was the only device at the time to run the OS. In November 2010 with the launch of the fourth generation of the OS, Apple renamed it iOS to coincide with the launch of the original iPad. This version was named iOS4 and there has been a new version each year culminating in the current release, iOS8, that launched in September 2014.

Where iOS7 was touted as a major UI refresh the focus of the features in iOS8 are the new frameworks and services as well as the newest device, Apple Watch. Tighter integration with iOS devices and Macs running Yosemite allow users to start tasks like creating emails on one device and finish them on another via a concept called Handoff. Some of the new frameworks are HealthKit which allows users to manage health-related information, HomeKit for communicating and controlling devices in a user's home, and CloudKit that provides a conduit for moving data between your app and iCloud. This just scratches the surface of all the new frameworks and APIs available in iOS8. For a complete list read "What's New in iOS" in Apple's iOS Developer Library⁴.

³ www.nytimes.com/2013/10/06/magazine/and-then-steve-said-let-there-be-an-iphone.html

⁴ developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS8

iOS Install Base

In addition to selling over 800 million iOS devices, a plus in Apple's favour is the high adoption rate of each iOS version soon after release. This allows developers to focus on the latest version as a development target and not worry about supporting a lot of devices on older versions, which has been a challenge for Android developers. Three weeks after the launch of iOS 8 Apple announced it had captured 48% of all iOS devices⁵ with 46% still on iOS7, leaving only 6% of devices running an older iOS version. Contrast this with Android's OS version KitKat, which needed over a year to capture 34% of Android devices⁶.

Devices Running iOS

Instead of listing every device Apple has created that runs iOS, here are the current devices that support iOS 7 and 8 as these would be what a new developer today should target. iOS 8 supports all of these devices except for the iPhone 4:

- iPhone - 4, 4S, 5, 5C, 5S, 6, 6 Plus
- iPod Touch - 5th Generation, 4" screen size
- iPad - 2, 3rd Generation, 4th Generation, Air, Air 2
- iPad Mini - 1st Generation, Mini 2, Mini 3
- Apple TV (SDK only available to select 3rd party Developers)
- Apple Watch

A detailed list of iOS devices, their capabilities and supported iOS versions can be found on Wikipedia⁷.

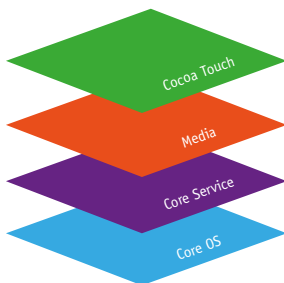
⁵ www.cnet.com/news/apples-ios-8-1-hits-monday-with-apple-pay-icloud-photo-library

⁶ developer.android.com/about/dashboards

⁷ en.wikipedia.org/wiki/List_of_iOS_devices

iOS Application Fundamentals

Like most Operating Systems the iOS Architecture is defined by layers of technologies to allow your application to run on a device without communicating directly at the hardware level. These technologies can be thought of layers or interfaces that are packaged as frameworks that the developer imports into their iOS projects to leverage. The primary framework developers interact with is called Cocoa Touch.



Cocoa Touch

While OSX and iOS are different operating systems, they share a lot in common in terms of frameworks, developer tools, and design patterns.

Apple leveraged and extended the main framework for developing OSX apps, Cocoa, and added support for unique features in iOS such as Touch gestures and called it Cocoa Touch. Included in Cocoa Touch are frameworks to build GUI interfaces, access device sensors like the accelerometer and perform networking and data management tasks.

The App Store

The primary method for deploying apps to consumers is through the App Store. Each app submitted is reviewed by the Apple review team to ensure it meets the requirements and standards set by Apple. This is a major difference from the Google Play store for Android apps where Google does not review apps but ensures they are code signed.

Apple is very strict on how 3rd party applications run on iOS and uses the Sandbox technique to ensure application security and prevent nefarious or buggy code that could compromise the OS, other applications or the device. Think of a sandbox as a virtual barrier around the application that sets the rules of what resources the app can access. For example an application does not have access to another app's file directory or system resources not accessed by the SDK frameworks. Apple has given more control to the user to grant access to their data (i.e. contacts, calendars, photos) or GPS location. Developers must prepare for cases where the user has denied these type of requests.

Getting Started with iOS Development

Mobile applications are commonly designated as being a "native" "mobile web" or "hybrid" app. Generally a native iOS App is built using Apple's platform, whereas a Hybrid app uses a 3rd party platform like Xamarin, Appcelerator and Phone Gap. These platforms try to make developing for multiple mobile platforms possible using one set of tools and language. Mobile Web apps typically use HTML5 standards to create what looks like a native app via a web browser on the device.

Along with the SDK to develop for iOS, Apple also provides an Integrated Development Environment (IDE) called Xcode to create both iOS and OSX applications. As Xcode has evolved, Apple has strived to provide all the needed tools to write, test, monitor performance and deploy apps to the App Store all from inside Xcode.

Xcode

Apple released Xcode in 2003 for writing applications in OSX. Version 3 of Xcode supported the first iPhone SDK in 2008 and the most recent version is Xcode 6, released with iOS 8 in September 2014. Xcode is a set of development, performance testing and measuring tools used during the whole application development life-cycle. Interface Builder is a visual design tool used to design and wire together views of the application without writing code and is integrated with Xcode. Also provided is an iOS Simulator to allow developers to test their apps on all current devices without having to always install apps on physical devices.

Interface Builder

A lot of discussion in iOS developer circles is whether it is better to use Interface Builder to visually design the UI and application flow or to undertake it all manually with code. In the past this may have been a personal preference but with new devices and screen sizes like the Apple Watch and iPhone 6, the case can be made that Interface Builder is becoming more essential. One of the primary differences between iOS and Android development was not having to develop for several device types and screen sizes. However this line is becoming more blurry with iOS8 supporting five different screen sizes. Instead of supporting all 5 separately in your applications, Interface Builder uses concepts such as Auto-Layout and Adaptive Layout to aid the developer in supporting all screen sizes more easily. With each new version of Xcode, Interface Builder has seen improvement and advancement so it's apparent that Apple prefers developers to take advantage of it. Something a new iOS developer should consider.

Objective C

Objective C has its roots in the NeXTSTEP operating system developed in the 1980s from where OSX and iOS are derived. It is an object-oriented programming language that adds messaging to the C Programming language⁸. In fact C and C++ can be written alongside Objective C and some of the iOS frameworks only provide a C level API to access. However it has been criticized for having a quirky syntax with a plethora of asterisks, '@' signs, and square brackets which leads to a higher learning curve for developers coming from modern languages such as Java or C#. Incremental improvements have been added over the years including dot notation of object properties, blocks, collection literals and memory management via Automatic Reference Counting (ARC). But the remaining need to use pointers, header files and remain tightly coupled to the limitations and risks of the C language has left Apple to conclude a new modern language is needed.

Swift

In July 2010 Chris Lattner, Senior Director and Architect in the Developer Tools Department at Apple began implementing the basic language structure of a new programming language whose existence only a few people knew of. It became a major focus for the Apple Developer Tools group in July 2013 and almost a year later at Apple's World Wide Developer Conference (WWDC) Apple announced a new programming language for iOS and OSX called Swift. Lattner admits Swift is influenced by other languages such as C#, Ruby, Haskell, Python and countless others⁹

⁸ en.wikipedia.org/wiki/Objective-C

⁹ nondot.org/sabre

Apple felt the reason to create Swift was the need for modern language syntax that is more concise and easier to learn for new iOS developers, including modern features like inferred data types, data structure declarations, tuples, closures, optional semicolons and no pointers. It has been suggested Apple's support for Swift is to ensure iOS developers stay interested in Apple's tools and don't look at other platforms with modern language support for iOS development.

Performance Tools and Testing

In addition to providing the tools to develop iOS applications, Xcode also provides tools for performance monitoring and testing.

Instruments Instruments allows developers to collect data about the performance and behavior of their iOS apps over time. Some of the common templates offered allow developers to track memory leaks, or detect application "hot spots" using the profiler instrument. The Automation instrument is used to automate user interface tests in your iOS app through test scripts written by the developer. These scripts run outside of the app and simulate user interaction by calling the UI Automation API. It can be run on a device or simulator.

XC Test Framework XCTest is the test framework integrated with Xcode to provide extensive testing in an organized and efficient way. By default, new projects created in Xcode using one of the application templates will add a Test target to the project. This allows the developer to write their own unit test classes, execute them and analyze the results using the Test Navigator, all from inside Xcode.

Setting Up the Dev Environment

After registering for a free developer account at developer.apple.com access is granted to download Xcode, sample code, videos, and documentation. Requirements to run all Xcode tools is a Mac computer running OS X 10.8 (Mountain Lion) along with the iOS SDK. This setup will allow for the creation and testing of iOS apps to run in the iOS Simulator. To submit apps to the App Store you must upgrade the developer account at a cost of \$99 a year which also gives access to betas of future versions of Xcode and iOS as they are released.

Learning Resources

With the popularity of Apple's developer eco-system comes a multitude of learning resources in different formats to help a new developer start coding for iOS, and a lot of them are free. By taking advantage of these resources and others like them the learning curve of mastering iOS development will lessen considerably.

Websites and Blogs

- [Developer.Apple.com](https://developer.apple.com) contains complete reference and programming guides for developers to learn how to develop iOS apps and class reference of all classes in their public frameworks. The library website is organized by Resource Types, Topics, and Frameworks plus the ability to search. One important document to read before designing the first app to be submitted to the app store is the iOS Human Interface Guidelines¹⁰. It offers developers

¹⁰ developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG

recommendations on Apple approved ways to design apps to ensure a positive user experience. Violation of these recommendations will most likely lead to apps being rejected by the App Store during review for submission.

- RayWenderlich.com has become an essential site for free iOS tutorials written by his community of developers with the goal being "to take the coolest and most challenging topics and make them easy for everyone to learn - so we can all make amazing apps." The site has expanded into offering programming books and Video tutorials (with a paid membership). Subscribe to their weekly Podcast for the latest news relevant to developers and interviews with leaders in the iOS developer community.
- iOS.devtools.me is a website created by Adam Swinden that he updates daily with the best iOS developer tools and back-end services to help in developing apps. Content is organized by categories (i.e Design, Graphics, Debugging), most popular, and recently added. Also provided is a weekly newsletter on the latest additions to the site.
- iOSDevWeekly.com is a weekly round up of the best iOS development links every week. Dave Verwer operates the site and they offer a weekly email newsletter published every Friday.
- Galloway.me.uk, a blog by London based iOS Developer/Author Matt Galloway. His "Effective Objective-C 2.0"¹¹ is highly recommended when ready to start learning advanced features and tips on the language.
- Merowing.info is a blog from developer/trainer/speaker Krzysztof Zablocki who offers tutorials and insights into iOS development from his experience as a consultant. He

¹¹ available via amazon.com/Effective-Objective-C-2-0-Specific-Development/dp/0321917014

also is active in the Open Source Community creating tool and libraries for iOS developers.

- AshFurrow.com is another popular iOS blogger/developer who proudly states the purpose of this blog is for "Exploring the Pain Points of iOS." He has authored multiple iOS Development books, is an active speaker and involved in the Open Source Community.

Online Video Training

As a member of Apple's Developer program you get free access to all of Apple's World Wide Developer Conference videos, source code, and presentation files are available to download and stream via the website or WWDC iOS app from the past several years. Apple usually makes the videos available the day after the presentation where as previously it would take weeks to be available after each year's conference.

Lynda.com currently offers over 30 video courses with paid membership subscription for beginning iOS Development including courses for iOS8. Source code for the projects are available to download depending on the membership level chosen. They also have a free app in the App Store to watch videos on iOS devices.

One popular free resource for video training is offered by iTunes University of a full semester course taught at Stanford University on beginning iOS development. The lectures dive deep into the Objective-C language and iOS Frameworks. Viewers can even download the coding assignments. Videos are viewed through iTunes or the iTunesU app for iOS devices

Finally YouTube has quite a few free videos for Learning iOS development including a channel created by Mohammad Azam¹² that lists several screencast tutorials for iOS.

¹² www.youtube.com/user/azamsharp

Twitter iOS Power User List

Some of the biggest iOS developer experts are active on Twitter and provide great insight on latest trends and discussions on app development as well as helpful to answer developer questions. Recommended to start following and interact with:

- [@rwenderlich](#) - creator and Editor-in-Chief of [RayWenderlich.com](#)
- [@nicklockwood](#) - author of iOS Core Animation and is active in open source frameworks and apps
- [@NatashaTheRobot](#) - author of weekly Swift Newsletter "This Week in Swift"¹³ and iOS open source contributor
- [@tapbot_paul](#) - founder of [tapbots.com](#) and developer of TweetBot App.
- [@InvalidName](#) - author of multiple books, speaker, and developer
- [@casademora](#) - author, speaker, and developer
- [@jaredsinclair](#) - developer of Unread and other apps
- [@Cocoanetics](#) - developer, trainer, and speaker
- [@mattjgalloway](#) - author and developer
- [@azamsharp](#) - developer, author, podcaster, and creator of AzamSharp YouTube channel

Final Thoughts

It is an exciting time to be part of the iOS Development community and hopefully this chapter will prove helpful in finding a starting point. To say things change quickly is an understatement with all the new devices, frameworks, and services that have been launched in the past few years. But do

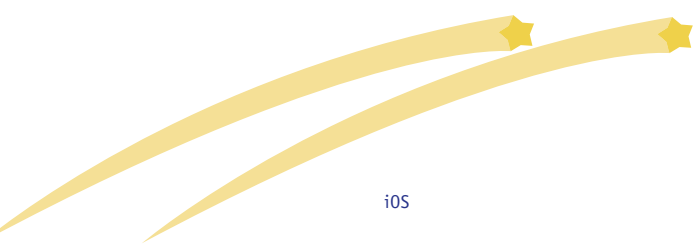
¹³ [swiftnews.curated.com](#)

not get intimidated by the speed at which technology moves inside Apple's eco-system. Most of the basics in developing a standard App apply now as they did in the first few versions of iOS. Luckily there are countless resources available to get started and grow your iOS development skills and most of them are free.

Things to consider when getting started on your first iOS "Hello World" App and beyond.

- Should I start with Objective-C or Swift language?
- Does it make more sense to use Interface Builder for designing the UI layout or to do it in code?
- While using back-end services like CloudKit make development easier am I locking myself too much into Apple's architecture which would make developing an Android version accessing same back end not possible?
- What are the drawbacks of developing iOS apps outside of Xcode (using cross-platform tools)? Is the user community big enough to find answers to issues? How well do their products keep up with the latest releases to iOS?
- What is the environment like today for being a full time iOS Indie Developer?

Answers to these questions are beyond the scope of the chapter but the resources above can help you navigate around the pitfalls in iOS development quicker on the way to being an experienced iOS Developer. Good luck and welcome to the club.



Java ME



Java ME (J2ME)

The Ecosystem

J2ME (officially "Java ME") is the oldest mobile application platform still widely used. J2ME is designed to run primarily on feature phones, and it dominates this market segment. However, feature phones are gradually being phased out by low-end smartphones which offer better hardware, APIs and monetization options at a similar price point. As a consequence, J2ME's popularity has declined significantly in recent years, and continues to do so.

So why would you want to develop for J2ME? Mainly for one reason: market reach. Although 72% of all mobile phones shipped in Q3 2014 were smartphones¹, most mobile phones in use today are feature phones. So if your business model relies on access to as many potential customers as possible, then J2ME might still be a great choice - especially when you are targeting markets like certain African countries or India.

However, if your business model relies on direct application sales, or if your application needs to make use of state-of-the-art features and hardware, smartphone platforms are the better choice.

¹ counterpointresearch.com/marketmonitor2014q3

Prerequisites

To develop a Java ME application, you will need:

- The Java SDK² and an IDE such as Eclipse Pulsar for Mobile Developers³, NetBeans⁴ with its Java ME plug-in or IntelliJ⁵. Beginners often chose NetBeans.
- An emulator, such as the Wireless Toolkit⁶, the Micro Emulator⁷ or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard⁸. For professional development, consider using a build tool such as Maven⁹ or Ant¹⁰.
- You may want to check out J2ME Polish¹¹, the open source framework for building your application for various devices.

Complete installation and setup instructions are beyond the scope of this guide, please refer to the respective tools' documentation.

Also download and read the JavaDocs for the most important technologies and APIs: you can download most Java-Docs from www.jcp.org. For manufacturer-specific APIs, documenta-

² oracle.com/technetwork/java/javame/downloads

³ eclipse.org

⁴ netbeans.org

⁵ jetbrains.com

⁶ oracle.com/technetwork/java/download-135801.html

⁷ code.google.com/p/microemu

⁸ proguard.sourceforge.net

⁹ maven.apache.org

¹⁰ ant.apache.org

¹¹ j2mepolish.org

tion is usually available on the vendor's website (for example, the Nokia Asha Documentation¹²).

Implementation

The Java ME platform comprises the Connected Limited Device Configuration (CLDC)¹³ and the Mobile Internet Device Profile (MIDP)¹⁴. As both CLDC and MIDP were designed a decade ago, the default set of capabilities they provide is rudimentary by today's standards.

Manufacturers can supplement these rudimentary capabilities by implementing various optional Java Specification Requests (JSRs), for example user data and file system access (JSR 75) or GPS support (JSR 179). For a comprehensive list of JSRs related to Java ME development, visit the Java Community Process' List by JCP Technology¹⁵.

It is very important to know that the JSRs you want to use may not be available for all devices; so capabilities available on one device might not be available on another device.

The Runtime Environment

J2ME applications are called MIDlets. A MIDlet's lifecycle is quite simple: it can only be started, paused and destroyed. On most devices, a MIDlet is automatically paused when minimized; it cannot run in the background. MIDlets also run in isolation from one another and are very limited in their interaction with the underlying operating system – these

¹² developer.nokia.com/asha/documentation

¹³ oracle.com/technetwork/java/overview-142076.html

¹⁴ oracle.com/technetwork/java/overview-140208.html

¹⁵ jcp.org/en/jsr/tech?listBy=1&listByType=platform

capabilities are provided strictly through optional JSRs (for example, JSR 75) and vendor-specific APIs.

Creating UIs

You can create the UI of your app in several ways:

1. **Highlevel LCDUI components:** you use standard UI components, such as Form and List
2. **Lowlevel LCDUI:** you manually control every pixel of your UI using low-level graphics functions
3. **SVG:** you draw the UI in scalable vector graphics then use the APIs of JSR 226¹⁶ or JSR 287¹⁷.

In addition, some manufacturers provide additional UI extensions. For example, Nokia Asha devices have their own specific UI paradigm¹⁸.

There are also tools that can help you with the UI development:

1. **J2ME Polish¹⁹:** This tool separates the design in CSS and you can use HTML for the user interface. It is backward compatible with the highlevel LCDUI framework
2. **LWUIT²⁰:** A Swing inspired UI framework
3. **Mewt²¹:** Uses XML to define the UI

Screen resolutions for Java ME range from 176x208/220 to

¹⁶ www.jcp.org/en/jsr/detail?id=226

¹⁷ jcp.org/en/jsr/detail?id=287

¹⁸ developer.nokia.com/asha/learning/ui

¹⁹ j2mepolish.org

²⁰ lwuit.java.net/

²¹ mewt.sourceforge.net

360x640, with the most popular being 240x320. Handling so many different resolutions can be a challenge, but using the above tools you can create UI layouts that scale automatically. Creating custom UIs for each resolution is possible, but not recommended: it is time consuming, error prone and expensive.

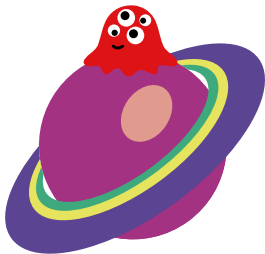
Graphical assets should always be optimized. A great free tool for this is PNGGauntlet²².

Testing

Because of device fragmentation, testing applications is vital. Test as early and as often as you can on a mix of devices. Some emulators are quite good, but there are some things that have to be tested on devices. Vendors like Nokia²³ and Samsung²⁴ provide subsidized or even free remote access to selected devices.

Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit²⁵, MoMEUnit²⁶ and CLDC Unit²⁷; Advanced tools like JInjector²⁸ provide code-coverage and UI testing support.



²² pnggauntlet.com

²³ forum.nokia.com/rda

²⁴ developer.samsung.com

²⁵ j2meunit.sourceforge.net

²⁶ momeunit.sourceforge.net

²⁷ snapshot.pyx4me.com/pyx4me-cldcunit

²⁸ code.google.com/p/jinjector

Porting

At its core, Java ME is a set of standards and specifications, which vendors sometimes interpret differently. This results in all kinds of bugs and non-standard behavior. In the following sections we outline different strategies for porting your applications to different Java ME handsets and platforms.

Lowest Common Denominator

You can prevent many porting issues by limiting the functionality of your application to the lowest common denominator. This usually means CLDC 1.0 and MIDP 1.0, or CLDC 1.1 and MIDP 2.0 if you only plan to release your application in more developed countries / regions.

While this approach is good for simple applications, comprehensive and feature-rich applications will be limited by it. In this case, you might want to consider using Java Technology for the Wireless Industry (JTWI, JSR 185) or the Mobile Service Architecture (MSA, JSR 248) as your baseline, but be aware that these have more limited support in the market.

Porting Frameworks

Porting frameworks help you deal with fragmentation by automatically adapting your application to different devices and platforms. To achieve this, they provide specialized run-time client libraries and built-time tools (such as cross compilers) that work together to make the process almost seamless.

Good porting frameworks enable you to use platform and device specific code in your projects. In other words: a good framework does not hide device fragmentation, but makes it more manageable.

For Java ME one of your options is J2ME Polish from Enough Software²⁹ (available under both the GPL Open Source license and a commercial license). Porting from C++ to Java ME is also possible with the open source MoSync SDK³⁰.

For more information about cross-platform development and the available toolsets, please see the “Going Cross-Platform” chapter.

Signing

The Java standard for mobile devices differentiates between signed and unsigned applications. Some handset functionality is available to trusted applications only.

Applications signed by the manufacturer or carrier of a device enjoy the highest security level and can access every Java API available on the handset.

Applications signed by JavaVerified³¹ are on a lower security level, while unsigned applications are on the lowest security level.

Which features are affected and what happens if the application is not signed is largely dependent on the implementation. Furthermore, not every phone carries all the necessary root certificates. The result is something of a mess, so consider signing your application only when required. In some cases an app store may offer to undertake the signing for you.

Another option is to consider using a testing and certification service provider and leaving the complexity to them. Intertek³² is probably the largest such supplier.

²⁹ enough.de

³⁰ mosync.com

³¹ javaverified.com

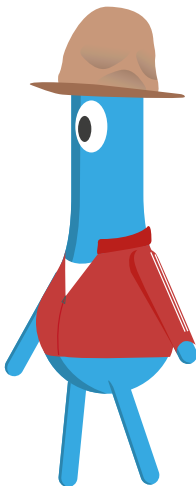
³² intertek.com/wireless-mobile

Distribution

App stores are probably the most efficient way to distribute your apps. Some of the most effective stores include:

- Mobile Rated³³ is a carrier and vendor independent application store.
- GetJar³⁴ is one of the oldest distributors for free mobile applications - not only Java applications.
- Opera Store³⁵, replacing Nokia's appstore which will be retired during 2015, migration of content started in January
- Carriers are in the game also, such as O2³⁶.

An overview of the available app stores (not those selling J2ME apps alone) can be found in the WIP App Store Catalogue³⁷. Also see the separate chapter on Appstores in this guide to learn more.



³³ mobilerated.com

³⁴ getjar.com

³⁵ apps.opera.com

³⁶ mobileapps.o2online.de

³⁷ wial.com/appstores

Learn More

If you want to learn more about Java ME development, below are a few resources that might help you.

Online

As Java ME is one of the oldest mobile platforms still used, it is easy to find resources related to it. For example:

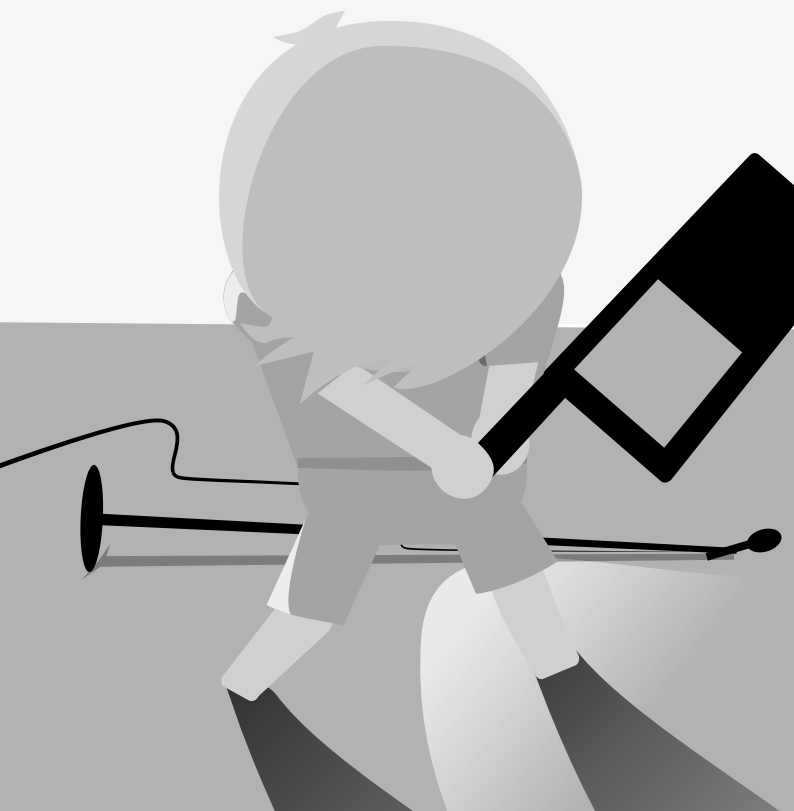
- Tutorials from sites such as J2MESalsa.com.
- Resource archives from sites such as billday.com/javame

Books

Over the years, a number of good Java ME books have been written, for example:

- **Beginning J2ME: From Novice to Professional** by Jonathan Knudsen and Sing Li
- **Pro Java Me Apps: Building Commercial Quality Java ME Apps** by Ovidiu Iliescu
- **Pro J2ME Polish: Open Source Wireless Java Tools Suite** by Robert Virkus, dealing with J2ME Polish development.
- **LWUIT 1.1 for Java ME Developers** by Biswajit Sarkar, dealing with LWUIT development

Unfortunately, due to Java ME's decreasing popularity, very few Java ME books have been written in recent years.



Tizen

The Ecosystem

Tizen is an open source, Linux based operating system designed to run on smartphones, netbooks, In-Vehicle-Infotainment (IVI) systems, Smartwatches, Smart-TVs and other 'smart' devices. It can be seen as a successor to Nokia/Intel's Meego and Samsung's LiMo, which were earlier smartphone operating systems based on Linux. Samsung merged the remains of their abandoned bada OS into Tizen, providing a framework for native apps. Tizen, as a brand of the Linux Foundation, was first announced by the Tizen Association in December 2011 and version 1.0 (codename 'Larkspur') was released in April 2012. Since then, the system has been under continuous development, with Tizen 3.0 due for releases in 2015. The main drivers of Tizen are Samsung for the mobile branch and Intel for IVI. Examples of other contributing companies are Fujitsu, NTT Docomo, Huawei, Vodafone and Orange.

The first Tizen device that became publicly available was Samsung's camera NX300M. On the Mobile World Congress in February 2014 Samsung then released new versions of their Smartwatches running on Tizen: the Samsung Gear 2 and Gear 2 Neo. Almost one year later the first Tizen smartphone finally became available in India and Samsung announced that all their Smart TVs released during 2015 will run Tizen.

Development

The main focus of Tizen is to enable a standards-based operating system for running apps written in HTML5 which is backed by the Tizen web browser currently holding the record for the best HTML5 implementation on mobile devices. Tizen also enables you to write native apps in C++, giving the power to max out the capabilities of hardware. Both development paths are supported by a variety of popular frameworks and libraries, such as JQuery, to give you a good start with your first Tizen app.

Web app developers can use a comprehensive list of HTML5 features, Tizen device APIs and libraries - such as JQuery and JQuery Mobile — to create beautiful apps. If you have created web apps for bada, you can use the most of the original code on Tizen.

The official Tizen SDK contains an Eclipse-based IDE, which can be used for both web and native app development. Former bada developers will recognize the roots of this SDK, Samsung's bada SDK. A code editor, UI designer and device emulator are all there ready to go. For web-based applications you can also use Intel XDK if you prefer.

Changes in Version 2.3

After almost a year of stagnation, Tizen 2.3 was been released in November 2014 and it brought substantial differences, especially for native app developers. Until then, app developers were able to build apps on a C++ framework called OSP which was already used by Samsung for their bada Smartphones. This made it easy for bada developers to port their apps to Tizen. In version 2.3 OSP has disappeared and developers have to use *EFL* for native apps. There are rumors that a OSP compatibility

runtime library will be released soon in order to allow all those apps based on OSP to run in Tizen 2.3 or newer.

One of the main changes for web-app development in Version 2.3 is the usage of TAU (Tizen Advanced UI) instead of jQuery (you can still use it if you want) and the possibility for developers to develop apps for smartphones and wearable devices with only one SDK.

Testing your apps

The best tests are those done on a device. At the time of writing your only option would be trying to get of one of Samsung's rare Tizen test devices, the RD-PQ and RD-210 but these devices are hard to find. Therefore you will probably have to use the simulator and emulator included in the Tizen SDK, while waiting for the first mass market Tizen phones to become available. The simulator provides a simple approach to testing web-based apps, but is limited in its features and cannot be compared to real-life testing on a device. The emulator is much nearer to device experience and can be used to test native apps. It is a virtual machine based on QEMU and running an image of a current Tizen mobile installation.

For those cases where a real device is needed, you can use the Samsung Remote Test Lab¹. These labs are situated around the world and give you the opportunity to remotely connect to devices from within your Tizen SDK.

¹ developer.samsung.com/remotetestlab

Distribution

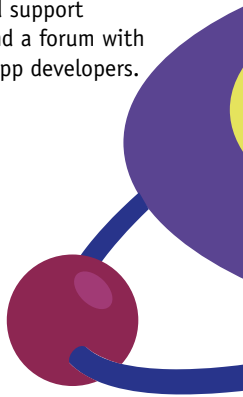
Tizen apps are distributed in Tizen package files (*.tpk) and widget files (*.wgt) created by the Tizen SDK. They are very similar to Android *.apk files, and can be installed by copying them to a Tizen device and opening them in a file explorer on the device. The main hub for distributing apps will be the TizenStore². In contrast to the Apple Appstore, Google Play Store and Microsoft Windows Phone Market you do not need to pay to become a registered developer at the Seller Office³. All submissions will be reviewed according to the Tizen Store guidelines. Usually, certification takes about 2 to 3 days, depending on the complexity of your app.

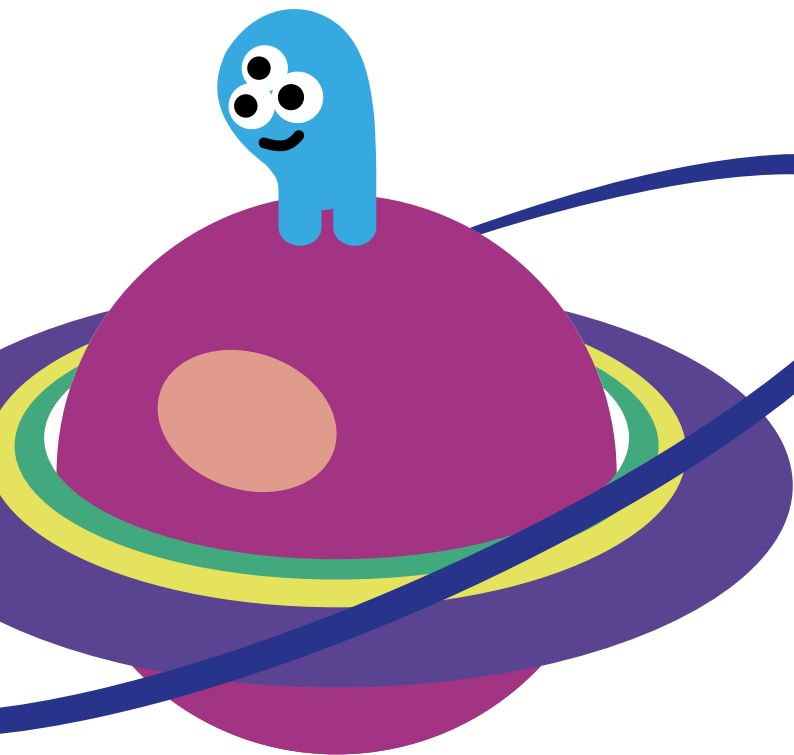
Learn More

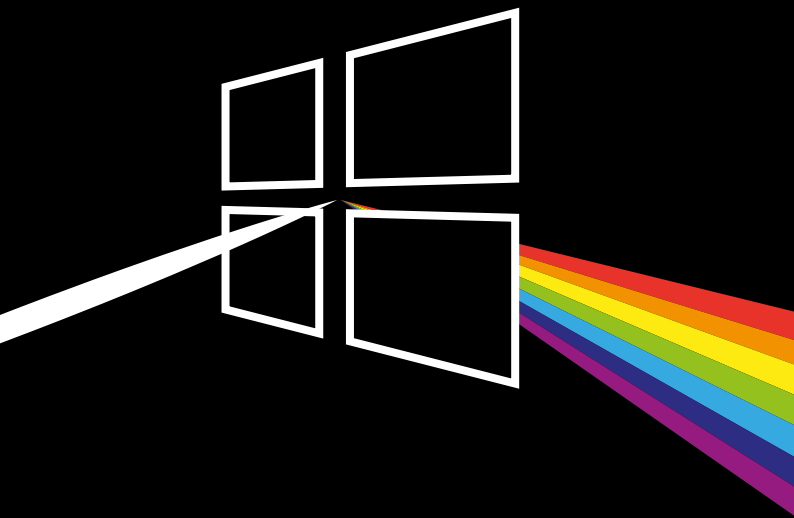
As a developer, your first stop should be developer.tizen.org. This site hosts all the documentation, tools and support services for Tizen development. You can also find a forum with a very active and friendly community of Tizen app developers.

² www.tizenstore.com

³ seller.tizenstore.com







Windows

With Windows 10 both the former Windows Phone and Windows PC platforms are going to converge to an even higher degree - along with an universal app store, universal app APIs and universal tooling. With Windows 10 this will not only span across phones, tablets and PCs, but you will also be able to target the TV via the Xbox One and even embedded devices. Windows 10 is already available as a developer's preview release and slated for release in mid 2015.

The Ecosystem

Microsoft changed the ecosystem in 2014 by buying the Nokia Devices and Services business, including the line of Lumia Windows Phones. Microsoft also removed any licensing costs associated with creating Windows phones or small Windows tablets. As a result a new range of devices have been announced and brought to market by regional players. If that will improve Windows' market share substantially remains to be seen. In 2014 the overall progress was far lower than it was hoped, but Microsoft was able to keep their second place after Android in some countries including Italy and Argentina¹. Windows also managed to reduce the app gap; by August 2014 there were more than 300,000 apps available².

Active Windows Phone vendors include: Microsoft, Samsung, Celkon Mobiles, Micromax, Lava International, Miia, Highscreen, Kazam, Blue, NGM, Yezz, Allview, Archos, HTC and others.

¹ kantarworldpanel.com/global/smartphone-os-market-share

² windowscentral.com/there-are-now-over-300000-windows-phone-apps-available

Languages and Tooling

Windows development is undertaken in C/C++, C# or VB.NET, using Microsoft Visual Studio IDE³. Applications are typically created using the WinRT XAML platform for event-driven applications, and DirectX, principally for games driven by a “game loop”. Both technologies can be used in a single application. Additionally you can create HTML 5 and JavaScript based apps using the corresponding WinRT JavaScript APIs, however web development is not covered in this chapter.

Last but not least you can create Windows apps without coding using the Windows Phone App Studio⁴ or the Project Siena⁵ app.

Thanks to Portable Class Libraries(PCL) and Windows Runtime Components you can use the best fitting language and UI framework for each module of your app.

If you want to use DirectX with C#, you can use SharpDX.org, anxframework.codeplex.com or game libraries based on that, such as monogame.codeplex.com.

While the most common scenario is to use XAML for apps and DirectX for games, you can also create XAML games and DirectX apps, depending on your needs. It is also possible to host Direct3D inside your XAML application. This could be used to display a 3D model inside an event-driven XAML application, or to easily create stylish Silverlight-based menus around a full DirectX game.

Historically Windows Phone apps were typically created with Silverlight, an app model that is very similar to the modern XAML approach. While the Store apps / WinRT based apps are

³ visualstudio.com

⁴ appstudio.windows.com

⁵ microsoft.com/projects/siena

the future, Silverlight still offered more integration options on Windows Phone 8.1.

Metro Design Paradigm

Microsoft pioneered the modern "flat" design paradigm that also influenced Android and iOS heavily. Its most obvious specific characteristic is the unique, simple-to-use interface that focuses on typography and content. This UI paradigm called Metro or Modern UI or Microsoft Design Language⁶ has been extended to the Xbox and Windows 8. This UI paradigm contains the following principles:

- **Content not Chrome** removes unnecessary ornaments and lets the content itself be the main focus. You should also refrain from using every available pixel, as whitespace gives balance and emphasis to content.
- **Alive in motion** adds depths to the otherwise flattened out design with rich animations
- **Typography is beautiful** moves fonts to first class citizens within Metro. The Helvetica inspired Segoe font of Windows Phone matches the modernist approach.
- **Authentically digital** design does not try to mimic real world object but instead focuses on the interactions that are available to digital solutions.

While Microsoft abandoned the 'Metro' name for its design paradigm due to legal worries, alternative names such as 'Modern UI' never really caught on.

You should embrace the Metro UI design principles in your application, especially when porting over existing apps. Designers will find many inspirations and information at design.windows.com.

⁶ [wikipedia.org/wiki/Metro_\(design_language\)](http://wikipedia.org/wiki/Metro_(design_language))

Important for the overall experience are the 'live tiles', small widgets that reside on the start screen. You can update them programmatically or even remotely using push notifications.

Integrating into the Platform

Numerous features are provided with the Windows Phone platform including lock screen, wallet, and location. Hardware, including sensors, Bluetooth, and proximity; and apps can use extensibility to embed the app into the Windows Phone experience⁷.

Apps can support 'live tiles' that show additional arbitrary information; the information can also be updated via push messages. Then you can use lockscreen apps that control the image of the lock screen on and integrate voice control using the available Cortana APIs.

In Windows Phone Silverlight apps you can create camera extension apps that are called lenses⁸ or integrate into the search experience⁹.

For a complete overview visit the integration documentation¹⁰.

In universal apps there are contracts¹¹ that serve the same purpose, you can handle specific file extensions, take part in sharing content and more.

⁷ Learn more at msdn.microsoft.com/en-us/library/windows/apps/hh202969

⁸ msdn.microsoft.com/library/windowsphone/develop/jj206990

⁹ msdn.microsoft.com/library/windowsphone/develop/hh202957

¹⁰ msdn.microsoft.com/library/windowsphone/develop/hh202969

¹¹ msdn.microsoft.com/library/windows/apps/hh464906

MVVM

For app developers coming from other platforms the data binding concepts of XAML will be new. For each page there should be a view model that includes the data for that page. The view itself only describes the UI, the display is populated with the data from the view model. Model classes contain the actual data. This concept of a Model, a View and a ViewModel (MVVM) ease the development of complex apps considerably.

Game Engines

Thanks to the native app capabilities there are various game engines available for Windows Phone 8 and Windows RT, for example: Cocos2d-x¹², Havok¹³, Marmalade¹⁴, OGRE¹⁵, Unity 3D¹⁶

Services

Push notifications¹⁷ are available that can also update the live tiles of your app. You can also consider using the freely available OneDrive cloud space and integrate with other Windows Live services¹⁸ in your app. There are many third party offerings¹⁹ available as well.

¹² cocos2d-x.org/wiki/Windows_Phone_8_Environment_Setup

¹³ havok.com/products/havok-windows-ecosystem

¹⁴ developer.madewithmarmalade.com/develop/supported-platforms

¹⁵ ogre3d.org/2012/10/30/ogre-now-supports-windows-phone-8

¹⁶ unity3d.com/pages/windows

¹⁷ msdn.microsoft.com/library/windows/apps/xaml/hh913756

¹⁸ msdn.microsoft.com/live

¹⁹ dev.windowsphone.com/en-us/featured/partners

Multitasking and Application Lifecycle

Windows has a limited form of multitasking that suspends applications in the background and allows for fast application switching. The only processes that can be run in the background, after an application has been left, are audio playback, location tracking and file transfer. Applications can also schedule to run arbitrary code in the background at an interval (code which is known as Background Tasks). Background Tasks are allowed limited use of resources and may be stopped or skipped if the OS determines that the phone needs to conserve resources.

Applications suspended in the background may be closed automatically if the OS determines resources are needed elsewhere.

To create the appearance of an application that was never closed, Windows has a well-documented application lifecycle called Tombstoning²⁰.

Testing and Analytics

Microsoft provides an excellent overview of testing a Windows Phone 8 app²¹ including tests throughout the development process.

Unit tests are integrated into Visual Studio, just create an additional Unit Test project in your solution and reference the projects you would like to test. MSDN includes a good overview on the process²²

²⁰ msdn.microsoft.com/library/windows/apps/xaml/hh464925

²¹ [msdn.microsoft.com/en-us/library/windows/apps/jj247547\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/jj247547(v=vs.105).aspx)

²² [msdn.microsoft.com/en-us/library/windows/apps/dn168930\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/dn168930(v=vs.105).aspx)

For behavior-driven development, the Windows Phone Test Framework by Expensify²³ is available.

For developers wishing to collect runtime data and analytics, there are several options. Flurry²⁴ and Google Analytics²⁵ provide analytics tools and services that are compatible with Windows. Developers can also use the Microsoft Application Insights service²⁶. There are robust performance monitoring tools available in Visual Studio for monitoring the performance during development.

Distribution and Monetization

Distribute your apps through the Windows Phone Store. While application content is reviewed and restricted in a way similar to the Apple App Store, Microsoft provides fairly comprehensive guidelines for submission, available at Dev Center²⁷. Although developer tools are provided free of charge, a paid Store account is necessary to deploy software to devices through the Windows Store. Currently, a developer account costs a once-in-a-lifetime fee of 19 USD for individuals and 99 USD for companies. The fee is waived for students in the DreamSpark²⁸ program. The Store also provides for time-limited beta distribution and offers a company hub for enterprises²⁹. You can use the Windows Certification Kit that are both integrated

²³ github.com/Expensify/WindowsPhoneTestFramework

²⁴ flurry.com/flurry-analytics.html

²⁵ googleanalyticsdk.codeplex.com

²⁶ azure.microsoft.com/documentation/articles/app-insights-get-started

²⁷ dev.windows.com

²⁸ www.dreamspark.com

²⁹ msdn.microsoft.com/library/windowsphone/develop/jj206943

with Visual Studio to test your application locally before you submit them.

There are 128 Regional Windows Phone App Stores supporting 50 languages³⁰, so you can have a global reach.

Apps are managed by customer, not by device. So a user can use your app across a variety of platforms, such as a desktop PC and a tablet. If you create a universal app you can choose if it only needs to be bought on one platform or if it needs to be bought again on each used platform.

For paid applications, the Windows Phone framework provides the ability to determine if your application is in "trial mode" or not and limit usage accordingly. Microsoft specifically recommends against limiting trials by time (such as a thirty-minute trial) and instead suggests limiting features instead³¹.

For ad-based monetization, there are several options. Microsoft has their own Microsoft Advertising Ad Control³² (currently available in 18 countries), Smaato³³, inneractive³⁴, AdDuplex³⁵ and Google³⁶ all offer alternative advertising solutions. You can also use the Ad Mediation³⁷ service for selecting different ad providers depending on the maximum achievable ad pricing. For more general information about monetization, please the dedicated chapter in this guide.

³⁰ kb.tethras.com/localizing-your-windows-phone-8-app/windows-phone-stores-and-supported-languages

³¹ msdn.microsoft.com/library/windowsphone/develop/ff967558

³² advertising.microsoft.com/mobile-apps

³³ smaato.com

³⁴ inner-active.com

³⁵ adduplex.com

³⁶ developers.google.com/mobile-ads-sdk/

³⁷ msdn.microsoft.com/library/windows/apps/xaml/dn864359

Learn More

Visit dev.windows.com for news, developer tools and forums.

The development team posts on their blog at blogs.windows.com/buildingapps or their Twitter account [@wpdev](https://twitter.com/wpdev). For a large collection of developer and designer resources, visit windowsphonegeek.com and reddit.com/r/wpdev.

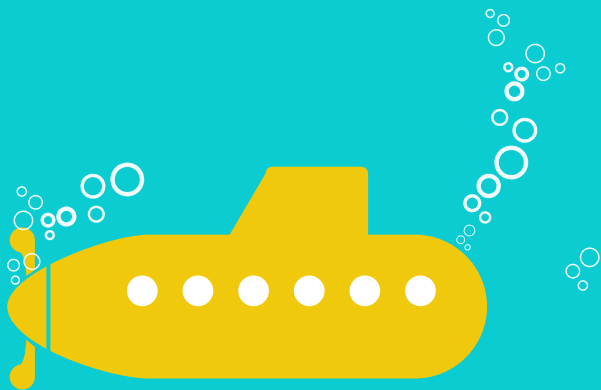
You can extend your components, behaviors and other tools with various commercial and open source toolkits. Popular examples include telerik.com, coding4fun.codeplex.com, cimbolino.org, mvvmlight.codeplex.com and github.com/MvvmCross/MvvmCross. For inspecting the visual tree, bindings and properties of XAML-based user interfaces at runtime, xamlspy.com is available.

Microsoft provides a series of videos for newcomers to developing Windows Phone 8.1 applications³⁸. And they publish many more videos at channel9.msdn.com.

Find sample code on code.msdn.microsoft.com/windowsapps, in various codeplex.com projects and in the end to end samples available at msdn.microsoft.com/library/windows/apps/br211375. The roadmap for app developers provides an good overview about planning, designing and developing Windows apps at msdn.microsoft.com/library/windows/apps/xaml/br229583.

If you are porting an existing app from iOS or Android you can find help at msdn.microsoft.com/library/windows/apps/dn751495.

³⁸ microsoftvirtualacademy.com/training-courses/wind1-development-for-absolute-beginners



Going Cross-Platform

So many platforms, so little time: This accurately sums up the situation that we have in the mobile space. There are more than enough platforms to choose from: Android, BlackBerry 10, Firefox OS, iOS, Tizen, Windows 8, and Windows Phone are or will likely be among the most important smartphone and tablet platforms while Brew MP and Java ME dominate on feature phones (arranged not by importance but rather alphabetically).

Most application sponsors, to quote Queen's famous lyrics, will tell the developer: "I want it all, I want it all, I want it all ...and I want it now!" So the choice may be between throwing money at multiple parallel development teams, or adopting a cross-platform strategy.

Key Differences Between Mobile Platforms

If you want to deliver your app across different platforms you have to overcome some obstacles. Some challenges are easier to overcome than others:

Programming Language

By now you will have noticed that most mobile platforms release their own SDKs, which enable you to develop apps in the platforms' supported programming languages.

However, these languages tend to belong to one of a few families of root languages and the following table provides an overview of these and the platforms they are supported on:

Language	1st Class Citizen ¹	2nd Class Citizen ²
ActionScript	BlackBerry 10, BlackBerry PlayBook OS (QNX)	none
C, C++	BlackBerry 10, Brew MP, Sailfish OS, Windows 10	Android (partially, using the NDK), iOS (partially)
C#	Windows 10	none
Java	Android, BlackBerry, Java ME devices	none
JavaScript	BlackBerry PlayBook OS, Firefox OS, Tizen, Windows 10	BlackBerry (WebWorks), Nokia (WRT)
Objective-C	iOS	none

- 1 Supported natively by the platform, for example either the primary or only language for creating applications
- 2 Supported as an option by the platform, for example can be used as an alternative to the native language but generally won't provide the same level of access to platform features.

Cross platform frameworks can overcome the programming language barriers in different ways:

- **Web Technologies:** This approach exploits the fact that most platforms provide direct support for web technologies through embedded 'webviews' in native applications. Along with HTML and CSS, this approach supports JavaScript also.

- **Interpretation:** Here the framework delivers an engine for each platform that interprets a common or framework specific language. For example, a popular option for games development is Lua scripting.
- **Cross Compilation:** The holy grail of cross platform frameworks is cross compilation, but it is also the most complex technical solution. It enables you to write an app in one language and have it transcoded into each platform's native language, offering native runtime speed.

Most frameworks also provide a set of cross platform APIs that enable you to access certain platform or device features, such as a device's geolocation capabilities, in a common way. For features such as SMS messaging you can also use network APIs that are device-independent¹.

OS Versions

Platforms evolve and sooner or later they will be version specific features that you want to leverage. This adds another layer of complexity to your app and also a challenge for cross-platform tools: sometimes they lag behind when a new OS version is released.

UI and UX

A difficult hurdle for the cross platform approach is created by the different User Interface (UI) and User eXperience (UX) patterns that prevail on individual platforms.

It is relatively easy to create a nice looking UI that works the same on several platforms. Such an approach, however, might miss important UI subtleties that are available on a single platform only and could improve the user experience

¹ www.developergarden.com/apis/

drastically. It would also ignore the differences when it comes to the platforms' design philosophies: While many platforms strive for a realistic design in which apps look like their real world counterparts, Windows Phone's Metro interface strives for an "authentically digital" experience, in which the content is emphasized not the chrome around it. Another key challenge with a uniform cross-platform UI is that it can behave differently to the native UI users are familiar with, resulting in your application failing to "work" for users. A simple example is not to support a hardware key such as the back key on a given platform correctly. Another challenge is the "uncanny valley" that results from mimicking native UI elements that look but do not work the same. Instead of mimicking native controls you should either use non-native looking ones or just use the 'real deal'.

When you target end consumers directly (B2C), you often need to take platform specific user experience much more into account than in cases when you target business users (B2B). In any case you should be aware that customizing and tailoring the UI and UX to each platform can be a large part of your application development effort and is arguably the most challenging aspect of a cross platform strategy.

Desktop Integration Support

Integration of your application into devices' desktops varies a lot between the platforms; on iOS you can only add a badge with a number to your app's icon, on Windows Phone you can create live tiles that add structured information to the desktop, while on Android you can add a full-blown desktop widget that may display arbitrary data and use any visuals.

Using desktop integration might improve the interaction with your users drastically.

Multitasking Support

Multitasking enables background services and several apps to run at the same time. Multitasking is another feature that is realized differently among operating systems. On Android, BlackBerry and Sailfish OS there are background services and you can run several apps at the same time; on Android it is not possible for the user to exit apps as this is handled automatically by the OS when resources run low. On iOS and Windows Phone we have a limited selection of background tasks that may continue to run after the app's exit. So if background services can improve your app's offering, you should evaluate cross platform strategies carefully to ensure it enables full access to the phone's capabilities in this regard.

Battery Consumption And Performance

Closely related to multitasking is the battery usage of your application.

While CPU power is roughly doubled every two years (Moore's law says that the number of transistors is doubled every 18 months), by contrast battery capacity is doubling only every seven years. This is why smartphones like to spend so much time on their charger. The closer you are to the platform in a crossplatform abstraction layer, the better you can control the battery consumption and performance of your app. As a rule of thumb, the longer your application needs to run in one go, the less abstraction you can afford.

Also some platforms have a great variety of performance, most notably Android - Android devices range from painfully slow to über-fast.

Push Services

Push services are a great way to give the appearance that your application is alive even when it is not running. In a chat application you can, for example, send incoming chat messages to the user using a push mechanism. The way push services work and the protocols they use, again, can be realized differently on each platform. The available data size, for example, ranges between 256 bytes on iOS and 8kb on BlackBerry. Service providers such as Urban Airship² support the delivery across a variety of platforms.

In App Purchase

In app purchase mechanisms enable you to sell services or goods from within your app. Needless to say that this works differently across platforms. See the monetization chapter for details.

In App Advertisement

There are different options for displaying advertisements within mobile apps, some are vendor independent third-party solutions. Platform specific advertisement services, however, offer better revenues and a better user experience. Again, these vendor services work differently between the platforms. The monetization chapter in this guide provides more information on this topic as well.

Cross-Platform Strategies

This section outlines some of the strategies you can employ to implement your apps on different platforms.

² urbanairship.com

Direct Support

You can support several platforms by having a specialized team for each and every target platform. While this can be resource intensive, it will most likely give you the best integration and user experience on each system. An easy entry route is to start with one platform and then progress to further platforms once your application proves itself in the real world.

Component libraries can help you to speed up native development, popular examples are listed in the following table.

Component Library	Target Platforms
cocoacontrols.com	iOS
chupamobile.com	Android, iOS
verious.com	Android, iOS, HTML5, Windows
windowsphonegeek.com/Marketplace	Windows

Asset Sharing

When you maintain several teams for different platforms you can still save a lot of effort when you share some application constructs:

- **Concept and assets:** Mostly you will do this automatically: share the ideas and concepts of the application, the UI flow, the input and output and the design and design assets of the app (but be aware of the need to support platform specific UI constructs).
- **Data structures and algorithms:** Go one step further by sharing data structures and algorithms among platforms.
- **Code sharing of the business model:** Using cross platform compilers you can also share the business model between the platforms. Alternatively you can use an

interpreter or a virtual machine and one common language across a variety of platforms.

- **Complete abstraction:** Some cross platform tools enable you to completely abstract the business model, view and control of your application for different platforms.

Player And Virtual Machines

Player concepts typically provide a common set of APIs across different platforms. Famous examples include Flash, Java ME and Lua. This approach makes development very easy. You are dependent, however, on the platform provider for new features and the challenge here is when those features are available on one platform only. Often player concepts tend to use a “least common denominator” approach to the offered features, to maintain commonality among implementations for various platforms. Generator concepts like Applause³ carry the player concept a step further, they are often domain specific and enable you to generate apps out of given data. They often lack flexibility compared to programmable solutions.

Cross Language Compilation

Cross language compilation enables coding in one language that is then transformed into a different, platform specific language. In terms of performance this is often the best cross platform solution, however there might be performance differences when compared to native apps. This can be the case, for example, when certain programming constructs cannot be translated from the source to the target language optimally.

There are three common approaches to cross language compilation: direct source to source translation, indirectly

³ [applause.github.com](https://github.com/applause)

by translating the source code into an intermediate abstract language and direct compilation into a platform's binary format. The indirect approach typically produces less readable code. This is a potential issue when you would like to continue the development on the target platform and use the translated source code as a starting point.

(Hybrid) Web Apps

Some of the available web application frameworks are listed in the following table. With these frameworks you can create web apps that behave almost like real apps, including offline capabilities. However, be aware that the technologies have limitations when it comes to platform integration, performance, and other aspects. Read the web chapter to learn more about mobile web development.

Web App Solution	License	Target Platforms
Chrome Apps developer.chrome.com/apps	BSD	Android, Mac, Windows
jQuery Mobile	MIT and GPL	Android, BlackBerry, Firefox, iOS, Windows
Sencha Touch www.sencha.com/products/touch	GPL	Android, BlackBerry, iOS, Windows Phone
The M Project	MIT and GPL	Android, BlackBerry, Firefox, iOS, Windows

Typically you have no access to hardware features and native UI elements, so in our opinion they do not count as “real” cross platform solutions: these solutions are therefore not listed in the table at the end of this chapter.

Hybrid web development means to embed a webview within a native app. This approach allows you to access native

functionality from within the web parts of your apps and you can also use native code for performance or user experience critical aspects of your app. Hybrid apps allow you to reuse the web development parts across your chosen platforms - a well known example for a hybrid web framework is PhoneGap.

ANSI C

While HTML and web programming starts from a very high abstraction you can choose the opposite route using ANSI C. You can run ANSI C code on all important platforms like Android, BlackBerry 10, iOS and Windows 8/Windows Phone. The main problem with this approach is that you cannot access platform specific APIs or even UI controls from within ANSI C. Using C is mostly relevant for complex algorithms such as audio encoders. The corresponding libraries can then be used in each app project for a platform.

Cross-Platform App Frameworks

There are many cross-platform solutions available, so it is hard to provide a complete overview. You may call this fragmentation, we call it competition. A word of warning: we do not know about all solutions here, if you happen to have a solution on your own that is publicly available, please let us know about it at [. A framework needs to support at least two mobile platforms to be listed.](#)

Here are some questions that you should ask when evaluating cross platform tools. Not all of them might be relevant to you, so weight the options appropriately. First have a detailed look at your application idea, the content, your target audience and target platforms. You should also take the competition on the various platforms, your marketing budget and the know-how of your development team into account.

- How does your cross platform tool chain work? What programming language and what API can I use?
- Can I access platform specific functionality? If so, how?
- Can I use native UI components? If so, how?
- Can I use a platform specific build as the basis for my own ongoing development? What does the translated/generated source code look like?
- Is there desktop integration available?
- Can I control multitasking? Are there background services?
- How does the solution work with push services?
- How can I use in app purchasing and in-app advertisement?
- How does the framework keep up with new OS releases?

For a benchmark of the available frameworks refer to the research2guidance report available at research2guidance.com/cross-platform-tool-benchmarking-2014.

Solution	License	Input	Output
Adobe Air adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, BlackBerry Tablet OS, iOS, PC
Akula	Commercial	(Visual)	Android, BlackBerry, iOS, Windows Phone
Application Craft applicationcraft.com	Commercial	HTML, CSS, JavaScript	Android, BlackBerry 10, iOS, Windows Phone, mobile sites

Solution	License	Input	Output
Codename One	Commercial	Java	Android, BlackBerry, iOS, J2ME, Windows
Corona coronalabs.com (Corona Labs)	Commercial	JavaScript	Android, iOS, Kindle, Nook, Windows
Evthings Studio	Open Source	HTML, CSS, JavaScript	Android, iOS
Feedhenry feedhenry.com	Commercial	HTML, CSS, JavaScript	Android, iOS, HTML5, Windows
J2ME Polish (Enough Software)	Open Source + Commercial	Java ME, HTML, CSS	Android, BlackBerry, J2ME, PC
Kony One kony.com/products/apps	Commercial	HTML, CSS, JavaScript, RSS	Android, BlackBerry, iOS, J2ME, Windows, PC, Web
LiveCode (RunRev)	Commercial	English-like	Android, iOS, Windows and Web
M2Active service2media.com (Service2Media)	Commercial	Drag and Drop + Lua	Android, BlackBerry, iOS, Windows
MobiForms (MobiForms)	Commercial	Drag and Drop + MobiScript	Android, iOS, PC, Windows Mobile

Solution	License	Input	Output
NeoMAD neomades.com	Commercial	Java	Android, BlackBerry, iOS, J2ME, Windows
Orubase	Commercial	ASP .NET MCV	Android, iOS, Windows
PhoneGap/Cordova phonegap.com cordova.apache.org (Adobe/Apache)	Open Source	HTML, CSS , JavaScript	Android, BlackBerry 10, iOS, Windows
Qt (Digia)	Open Source + Commercial	C++	Android, BlackBerry 10, iOS, Sailfish OS, Windows
Rhodes motorolasolutions.com/US-EN/RhoMobile+Suite/Rhodes (Motorola)	Open Source + Commercial	Ruby, HTML, CSS, JavaScript	Android, BlackBerry, iOS, Windows Phone
Titanium (Appcelerator)	Open Source	JavaScript	Android, iOS, Tizen, Windows, Mobile Web
trigger.io trigger.io (Trigger Corp)	Commercial	HTML5, JavaScript	Android, iOS
webinos	Open Source	JavaScript	Android, BlackBerry, iOS, PC, TV
Xamarin xamarin.com	Commercial	C#	iOS, Android, Windows, PC
XDK (Intel)	Free to use	HTML, CSS, JavaScript	Android, iOS, Windows

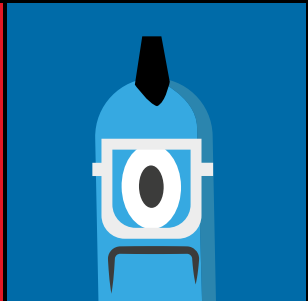
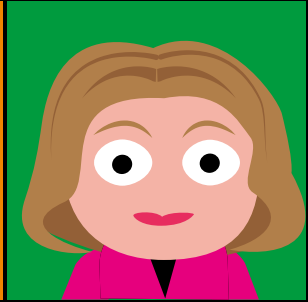
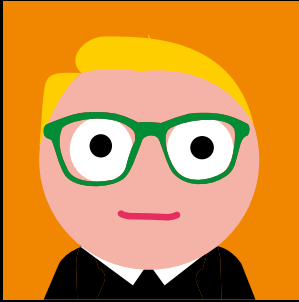
Solution	License	Input	Output
XML VM xmlvm.org	Open Source + Commercial	Java, .NET, Ruby	C++, Java, JavaScript, .NET, Objective-C, Python

Cross-Platform Game Engines

Games are very much content centric and often do not need to integrate deeply into the platform. So cross-platform development is often more attractive for games than for apps.

Solution	License	Input	Output
Cocos 2D cocos2d-x.org	Open Source	C++, HTML5, JavaScript	Android, BlackBerry, iOS, Windows
Corona (Corona Labs)	Commercial	Lua	Android, iOS, Kindle, nook, Windows
EDGELIB edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC
Esenthel (elements interactive)	Commercial	C++	Android, iOS, PC
GameSalad gamesalad.com	Commercial	Drag and drop	Android, iOS, Windows, PC, web
Gideros Mobile	Commercial	Lua	Android, iOS
id Tech 5 idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC

Solution	License	Input	Output
Irrlicht	Open Source	C++	Android & iOS with OpenGL-ES version, PC
Appease appeaseymobile.com	Open Source	C++	Android, iOS, Windows
Marmalade (Ideaworks3D)	Commercial	C++, HTML5, JavaScript	Android, BlackBerry 10, iOS, Windows
Moai getmoai.com (Zipline Games)	Open Source, Commercial	Lua	Android, iOS, PC, Web
MonoGame	Open Source	C#, XNA	Android, iOS, PC, Windows
Ogre 3D ogre3d.org	Open Source	C++	Windows Phone, Window RT, PC
orx	Open Source	C, C++, Objective-C	Android, iOS, PC
ShiVa 3D stonetrip.com	Commercial	C++	Android, BlackBerry 10, iOS, PC, Consoles
SI02 (sio2interactive)	Commercial	C, Lua	Android, iOS, PC
Unigine unigine.com (Unigine corp.)	Commercial	C++, UnigineScript	Android, iOS, PC, PS3
Unity3D (Unity Technologies)	Commercial	C#, JavaScript, Boo	Android, BlackBerry 10, iOS, Windows, PC, consoles, web



Mobile Sites & Web Technologies

The continuous development of web technology coupled with an increase in Internet-capable devices promises a great future for those catering to the ever-increasing mobile web audience. Global mobile Internet traffic is growing rapidly and has already surpassed 30%¹. The share of time spent browsing the Internet by device (mobile, tablet, desktop and Smart TV) varies greatly across the globe. While users in China spend 16% of their time browsing the Internet on a mobile device, US Americans spend 27% and Indians 70%. Most regions where mobile Internet traffic has already surpassed desktop Internet traffic are developing and emerging markets. While smartphone use is growing around the globe, there is still a huge opportunity to cater to a feature phone audience in developing and emerging markets. Take India as an example. According to Nielsen, only 18% of Indians own a smartphone. Nevertheless this equates to a whopping 225 million users². At the same time 68% own a feature phone of which 9.5% are Internet-enabled feature phones³. Accordingly, a basic mobile site could help you reach another 85 million users.

The most obvious use of web technologies is to build mobile sites and this is also the key focus of this chapter. Nevertheless, it is worth pointing out that web technologies are also heavily used within web and hybrid mobile apps, cross-platform solutions and even native app development

¹ gs.statcounter.com

² nielsen.com/ph/en/insights/news/2014/asian-mobile-consumers.html

³ discovermobilelife.com

(e.g. Firefox and Tizen). For more information on cross-platform development and Firefox and Tizen OS, check out the respective chapters in this guide.

One big advantage of web technologies is that they offer the easiest route into mobile development. Web technologies, such as HTML, CSS and JavaScript have been well developed for many years; however they remain, and will continue to be, the main drivers of mobile site development. Additionally, they are arguably easier to learn than some of the rather complex languages needed for native app development. Mobile websites and web apps make content accessible on almost any platform with less effort in comparison to native development for a number of platforms. This means mobile websites automatically have a wider reach. Accordingly, mobile web development not only saves development time and cost, but furthermore provides a time and cost-effective alternative when it comes to maintenance. And being independent of app stores allows you to offer any content you want quickly, and without having to align it to the app store's approval policy.

Nevertheless there are shortcomings. Web technologies struggle to match the level of deep platform integration and direct access to hardware features native app development can provide. Furthermore performance of web technologies is highly dependent on connectivity, large sites such as Facebook and LinkedIn experience memory issues and there is a lack of existing developer tools in comparison to developer tools available for native app development.

Monetization of mobile sites can prove tricky as well, since users expect to access mobile sites free of charge. The most common monetization tool for mobile sites is ad integration. Payment solutions for mobile sites are still in their early stages and tend to be rather challenging to implement. Existing app store monetization tools by contrast offer easy set-up and a high level of security for the end-user.

If monetization is one of the key requirements, a hybrid or web app strategy could prove to be a good compromise. In that case the key challenge is to combine the unique capabilities of native and web technologies to create a truly user-friendly product. In the cross-platform chapter of this book you will find a list of available frameworks to create hybrid apps.

As a guiding principle users should not be left frustrated and disappointed by being directed to a site which takes forever to load, triggers high data charges, or does not work at all. Instead the worst-case scenario should be that a user is taken to a site that is basic but still provides all relevant content. Key criteria to consider prior to any development are your target audience's device capabilities, browsing habits and bandwidth/data plans.

From a UX perspective, Google offers 10 best practices to drive conversion for SMBs⁴:

- **Be thumb friendly** - design your site so even large hands can easily interact with it
- **Design for visibility** - ensure your content can be read at arm's length
- **Simplify navigation** - clear navigation, hierarchy and vertical scrolling aid access to information
- **Make it accessible** - ideally, your mobile site should work across all mobile devices and all handset orientations
- **Make it easy to convert** - focus on information that will aid conversion
- **Make it local** - including functionality that helps people find and get to you

⁴ www.dudamobile.com/webinar/Google_DudaMobile_Webinar.pdf

- **Use mobile site redirects** - give users a choice to go back to the desktop site, but make it easy to return to the mobile site
- **Keep it quick** - help mobile users, design your site to load faster and make the copy easy to scan
- **Make it seamless** - bring as much of the functionality of your desktop site to mobile
- **Learn, listen and iterate** - good mobile sites are user-centric, meaning they're built with input from your audience.

Google has also rolled out changes to its mobile search results and has announced that it will penalize sites that are not in line with its recommendations. Have a look at Google's developer site⁵ for more up-to-date information on how to optimize your mobile site.

HTML5

The fifth version of the HTML standard promises to reproduce features previously available only with the help of proprietary technology. HTML5 is one of the key drivers that makes coders and decision-makers consider developing mobile sites and web apps instead of native applications. A look-and-feel close to that of apps combined with a single code base for a number of popular devices, the ability to access device hardware such as the camera and microphone, local data storage for offline availability and optimization based on screen size make HTML5 an appealing alternative to native app development.

However HTML5 relies on universal browser support which is currently lacking.

⁵ developers.google.com/webmasters/smartphone-sites

Ex-Facebook CTO Brent Taylor describes the situation as follows: 'There is rampant technology fragmentation across mobile browsers, so developers do not know which part of HTML5 they can use. HTML5 is promoted as a single standard, but it comes in different versions for every mobile device. Issues such as hardware acceleration and digital rights management are implemented inconsistently. That makes it hard for developers to write software that works on many different phone platforms and to reach a wide audience.'

Leading sites such as LinkedIn traded its mobile-web based apps in for a new set of native applications. Kiran Prasad, LinkedIn's senior director for mobile engineering decided to build both a HTML5 site and rich native apps. His reasoning is that HTML5 is ready and important for the business, but not supported by the ecosystem as it should be. 'There are tools, but they are at the beginning. People are just figuring out the basics.'⁶

For more info on browser compatibility, check out the HTML5Test online⁷. The site provides both an overview and in-depth analysis of which HTML5 features are supported by which browser. Facebook has also developed ringmark⁸ which tests web browsers for 3 rings, or levels, of support for HTML5 features which helps developers to quickly check the level of support of various mobile (and desktop) web browsers.

To wrap it up: Almost everyone in the mobile business agrees that HTML5 will succeed in the long run and especially last year has seen a rapid adoption of HTML5. ABI research estimates that mobile devices with HTML5-compatible browsers

⁶ venturebeat.com/2013/04/17/linkedin-mobile-web-breakup

⁷ html5test.com/results/mobile.html

⁸ rng.io

will total 2.1 billion worldwide by the end of 2016.⁹ Operating systems will gradually increase support for HTML5 features and browsers to increase overall adoption and speed. Open-source platforms such as the Firefox OS, Sailfish, Tizen and Ubuntu should also help to speed up adoption. The Developer Economics 2014 research¹⁰ already ranks HTML5 as the most popular developer language (47%) followed by Java (42%).

Fragmentation Needs Adaptation

The biggest challenge of mobile site development is fragmentation. In theory all internet-enabled devices can access any mobile site via a browser. The reality however is that developers need to adapt and optimize mobile site content to cater to the ever increasing number of browsers and devices with varying levels of software and hardware capabilities.

Broadly speaking there are two approaches to optimizing content for mobile devices: Client-Side and Server-Side Adaptation.

- Client-Side Adaptation makes use of a combination of CSS and JavaScript running on the device to deliver a mobile-friendly experience.
- Server-Side Adaptation makes use of the server to execute logic before content is passed on to the client.

The following section provides an overview of client-side and server-side techniques used to make mobile sites accessible for the majority of current and future Internet-enabled devices.

⁹ www.abiresearch.com/press/21-billion-html5-browsers-on-mobile-devices-by-2016

¹⁰ DeveloperEconomics.com




Client Side Adaptation

Responsive Web Design

Responsive Web Design has been a buzzword amongst marketers and web developers alike. In its simplest form responsive design consists of a flexible grid, flexible images and CSS media queries to cater to a number of screen resolutions or types of devices.

On its own this results in a device-sensitive experience for a limited range of devices and lacks sophisticated content adaptation. The same content is served to all devices. It is not advisable as a technique to deliver complex desktop and mobile sites.

Pro

- 
- Pure client side adaptation ensures no impact on the existing infrastructure
 - Automatic adjustment of content and layout possible

Con

- The same content available on the website will also be available on the mobile version (whether visible or not).
- The page weight of the site can have a significant impact in terms of performance on mobile devices
- It is a general approach instead of actual mobile-friendly device optimization (e.g. Top 5)

Progressive Enhancement

Progressive Enhancement has the capability to cater to the full spectrum of mobile devices. A single HTML page is sent to every device. JavaScript code is then used to progressively build up functionality to an optimal level for the particular device. As a mobile only solution the main drawback is performance. The progressive build-up takes time to execute and varies according to the device and network. As a desktop and mobile solution its main drawback is that a single HTML document is sent to all devices. A well-known framework that makes use of progressive enhancement is jQuery Mobile¹¹.

Pro

- Pure client side adaptation ensures no impact on the existing infrastructure
- Progressive adjustment of content, function and layout possible

Con

- A loss of control, since detection is handled by the browser
- Browser detection is still far from perfect
- Detection done on the client-side impacts overall performance of the site
- The same HTML page is served to all devices

¹¹ jquerymobile.com

Server-Side Adaptation

Device Databases

Device databases detect each device accessing the website and return a list of device capabilities to the server. This information is then used to serve a mobile site that caters to the device's capabilities. Server-side adaptation is one of the oldest and most reliable solutions. Popular device databases include WURFL¹² and DeviceAtlas¹³. The main drawback of device databases is that the majority is only available as part of a commercial license.

Pro

- Most commonly used solution (Google, Facebook, Amazon and alike)
- Maximum control
- Device optimization possible (for example to iPhone, Samsung Galaxy and alike)

Con

- Device Description Repositories are hardware focused
- Besides the data, a detection mechanism is needed (a simple 'User-Agent' matching does not work)

¹² wurfl.sourceforge.net

¹³ deviceatlas.com

Hybrid Adaptation - RESS

Truly the best of both worlds, the combination of client and server-side adaptation ensures high performance thanks to server-side adaptation and means that the capabilities sourced can be used to enrich the mobile experience on subsequent visits. This approach is best known as RESS - responsive web design with server-side components. Naturally, this approach is costly and therefore more common within larger organizations.

RESS/ Hybrid Adaptation solutions are available commercially from companies such as Sevenval¹⁴ and Netbiscuits¹⁵, and as community-backed cloud solutions, for example FITML¹⁶.

Better Data Input

With small, often on-screen, keyboards entering text can be cumbersome and time-consuming, particularly if the user has to enter numbers, email addresses or similar text. Thankfully developers can easily specify the expected type of input and smartphones will then display the most appropriate on-screen keyboard. mobileinputtypes.com provides various clear and concise examples.

Better Performance

Mobile users expect sites to load within 2-5 seconds. This is currently a challenging task, especially for complex mobile sites. Please note that the location and network used can already have a drastic effect on site performance. While there are factors that you simply can't influence, the following sections provides tips to reduce transfer size, content and HTTP requests to minimize load time and improve performance.

¹⁴ sevenval.com

¹⁵ netbiscuits.com

¹⁶ fitml.com

Reduce Transfer Size

Activate GZIP when you serve a site. Make use of image resizing and adjust the image quality according to network quality.

Reduce Content

Both site and asset loading becomes more and more important. Minifying assets such as JavaScript and CSS files can help to reduce overall asset load times. Multiple files of the same type are compressed into one and all whitespace is removed. Code becomes shorter, but still behaves in the same way. All this can result in a lower number of requests and ultimately a faster loading time.

At the same time it is important that the user understands what is going on. Accordingly, if content is loading, it's important that the user is aware of this and is not presented with a blank box or page. A smooth experience is paramount to any mobile experience and this includes the journey from site to content loading in the site and any animation surrounding it.

Reduce HTTP Requests

Inline images, scripts and styles, and add make use of Application Caching. Whenever possible, reduce the number of requests, file size and content. Key benefits are that scripts are delivered in a single request per page, HTTP round-trips are minimized and core scripts are stored in the application cache. The implementation will not affect reload and scripts are still cacheable publicly (CDN).

For more detailed tips around mobile web performance check out Roland Guelle's presentation on slideshare¹⁷.

¹⁷ www.slideshare.net/sevenval/mobile-web-performance-dwx13

Testing Web Technologies

How web technologies work in various mobile phones can be tested in several ways. The simplest way is to test the web site or web app in a variety of web browsers on mobile devices. These would include a mix of the most popular mobile web browsers, based for example on public data available online¹⁸. The set of devices can be refined by analyzing data from existing web logs and similar sources. Also, testing on various form-factors helps to expose layout and formatting issues.

In terms of automated testing, WebDriver¹⁹ is the predominant framework. There are two complementary approaches:

1. Automated testing using embedded WebView controls in Android and iOS
2. User-agent spoofing using Google Chrome or Mozilla Firefox configured to emulate various mobile web browsers

Both approaches have pros and cons:

- Embedded WebViews run on the target platform OS. They are likely to find many behavioral bugs. However the configuration is more involved and some platform OSs are not supported.
- Spoofing can fool web servers to treat the browser as if it came from any of a wide range of devices, including mobile browsers not available with the embedded WebView such as the Nokia Asha 201 phone. However the behavior and rendering is not realistic so many bugs will remain undetected, while other 'false positive' bugs will be found that do not occur on devices.

¹⁸ gs.statcounter.com

¹⁹ seleniumhq.org/projects/webdriver

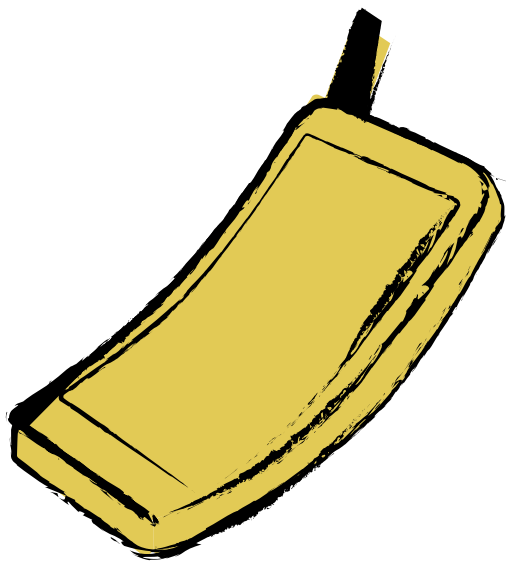
Learn More

Online

- **W3Schools and CSS Tricks** (good resource to understand basic HTML, CSS and JavaScript): w3schools.com, css-tricks.com
- **HTML5 Rocks** (great resource about HTML5 including tutorials, slideshows, articles and more): html5rocks.com/en
- **Breaking the Mobile Web** (Max Firtman, the author of several books about mobile web programming, provides up-to-date news in his dedicated mobile blog): mobilexweb.com
- **Mobi Thinking** (DotMobi's resource for marketers with insights, analysis and opinions from mobile marketing experts): mobithinking.com
- **Testing (Mobile) Web Apps:** docs.webplatform.org/wiki/tutorials/Testing_web_apps
- **Investigate what features work across all areas of the web:** caniuse.com and beta.theexpressiveweb.com
- **WHATWG** (The HTML community's homepage): whatwg.org
- **Word Wide Web Consortium** (The organization that defines web standards): w3.org

Books

- **Mobile First** by Luke Wroblewski
- **Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement** by Aaron Gustafson and Jeffrey Zeldman
- **Responsive Web Design** by Ethan Marcotte
- **Programming the Mobile Web** by Max Firtman
- **jQuery Mobile: Up and Running** by Max Firtman



Enterprise Apps

Corporate decision makers now view mobile enterprise apps as a strategic factor, a necessity, rather than an item on an accountant's spreadsheet. Internal enterprise apps are able to reduce the latency of information transfer within a company. They increase the agility of the worker by making competitive data available at any time and anywhere. Apps can also allow companies to engage with its customers, suppliers, and end consumers etc. Examples of enterprise apps include field & sales staff software, emergency response, inventory management, supply chain management but also B2C marketing.

It may seem an obvious thing to say, but the major risk at the moment, is **not** having an enterprise mobile strategy. Business is now looking at **Mobile for All** rather than limiting it to senior management, as it may have been in the past. To enable this the traditional IT approach of buying devices and distributing them throughout the management structure is no longer the only enabling strategy being used; we have moved from Bring Your Own Device (BYOD) to BYOx including apps, content, development tools/frameworks and now even wearables, enabling staff to use their personal devices to connect to the IT infrastructure, download secure content and use enterprise apps. With the advent of BYOx, a company exposes itself to risks which traditionally have never been part of the corporate IT strategy. Early adoption of a well thought out and implemented enterprise mobile strategy is key to ensuring data is secured at all times.

And from a developer's point of view, the enterprise sector has a lot to offer as well: Compared to traditional B2C app developers those who create enterprise apps are twice as likely to be earning over \$5k per app per month and nearly 3

times as likely to earn over \$25k according to the Developer Economics report¹.

Key points for Mobile Apps in Shaping the new Business Enterprise

- Cost reduction compared to existing systems
- Streamlining business processes
- Competitive advantage with up-to-date data immediately at hand
- Increase employee satisfaction and effectiveness
- Rapid response compared to existing processes

Enterprise Strategy

Many companies nowadays have a Chief Mobile Officer (CMoO) or have extended their CIO position. It is their job to coordinate mobile trends and directions and to bridge the gap between business and IT. Depending on the size and main focus of the company, his/her job is also to either build up an internal mobile software development team or coordinate the cooperation with an external development agency. To make sure that the mobile software delivers what the employees / users want, that this is technically achievable and that everything fits the overall company strategy, the leader might consider setting up a Mobile Innovation Council (MIC) or Center of Excellence (COE). This group should contain key members such as: skilled representatives from the mobile development team, stakeholders for mobile within the company, and most

¹ www.developereconomics.com/report/next-gold-rush-enterprise-apps

importantly end users from various departments with expertise in the relevant business processes.

Topics that the CMoO/CIO needs to focus on together with the MIC/COE include:

- **Strategy** - vision and direction for the general mobile strategy and for the apps.
- **Governance policies** - Bring Your Own Device (BYOD) vs. Chose Your Own Device (CYOD) which is essentially the difference between a Mobile Application Management (MAM) policy (BYOD) and a Mobile Device Management & Security (MDM) policy (CYOD)
- **App specifications**
- **App roadmap**
- **Budget planning**
- **Acceptance** - signing off the apps into production.
- **App deployment** - early feedback on demos and prototypes, testing, mass deployment
- **Incentives** - how to promote the adoption of mobile.

In commercial adoption terms Enterprise app development is more mainstream now, but one of the main hurdles a company writing third party enterprise apps, or a development manager keen to adopt an internal enterprise strategy will face is the requirement for a business need. The question “This all sounds great, but why do we need it?”, is less common now but you must be prepared to give compelling reasons for a company to adopt a mobile strategy.

Key points when building the business case for Mobile Enterprise Apps

- Create a Visionary Plan for more mobile Apps and know how they will aid and shape your enterprise
- Create an ADS (Application Definition Statement) for each App, specifying purpose and intended audience.
- Create a Budget for devices
- Create a plan for an Application & Device Management Strategy & Security Infrastructure.
- Create a plan for an App Dev Team using a future proof Development Platform - such as a MADP

Mobilizing Existing Systems

If you are already providing a system to customers which has not yet been mobilized, you will have various decisions to make. It is critical to fully understand the impact of adding a mobile offering to your system before you start implementation of the solution. Common reasons to mobilize your product can include using phone features, such as camera and GPS, or just the ability to capture information on the move, without being connected to the internet. You must ensure you go mobile for the right reasons, as the ongoing support, maintenance and development of a mobile offering will become a separate product roadmap to your original system and will carry an on-going cost.



Key points when deciding how to mobilize an existing system

- Clearly define the reasons for going mobile and ensure that those reasons are strong enough to take the step into mobile
- Understand the difference between mobile and desktop. Do not just copy your existing system, so for instance, instead of a form to capture information, you could capture audio and upload that into your system, allowing a user to quickly make notes without the need to type into a small device
- Do not try and implement all the features of your existing system; implement the important features in a way which suits mobile
- Ensure you understand which devices your clients use and which features of your system are most required to be mobilized
- Have a clearly defined mobile testing strategy which covers cross platform testing and multiple device types and operating systems

Device And Application Management In The Enterprise

When developing an enterprise app, you should always keep in mind that the hardware containing sensitive company data can get lost or stolen. There are now two approaches for securing devices, content and apps. Mobile Device Management (MDM) and Mobile Application Management (MAM). These are now coming together as Enterprise Mobility Management (EMM).

MDM gives an enterprise ultimate control over a device, so

when a device is lost, stolen or an employee leaves, taking the device, the enterprise can wipe the device and essentially stop the device from working. This approach is usually taken when an enterprise owns the device so all the data and apps on the device are owned by the company; any personal data stored on the device is stored at the employee's risk.

MAM enables an enterprise to adopt BYOD as it allows an enterprise to secure apps and content downloaded to a device without taking ultimate control away from the owner of the device. When an employee leaves a business, taking their device with them, the business can disable the enterprise apps and wipe any content downloaded to the device without affecting personal data, such as photos and consumer bought apps. Most MDM and MAM solutions are cross platform, supporting Apple, Android, Windows and BlackBerry devices, and this should always be taken into consideration when deciding upon an MDM or MAM provider.

Various security features are available through both these management solutions, including

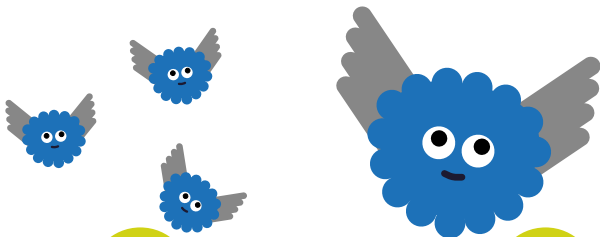
- Device monitoring
- License control
- Distribution via an internal Over-The-Air (OTA) solution
- Software inventory
- Asset control
- Remote control
- Connection management
- Application support & distribution

Security measurements include

- Password protection
- On-device data encryption
- OTA data encryption
- Remotely lock devices
- Remotely wipe data
- Re-provision devices
- Back-up data on devices

Examples of EMM providers are:

- **Airwatch:** air-watch.com
- **App47:** app47.com
- **Apperian:** apperian.com
- **Good:** good.com
- **Microsoft:** microsoft.com/en-us/windows/windowsintune
- **MobileIron:** mobileiron.com
- **Mocana:** mocana.com
- **SAP Afaria:** sap.com/pc/tech/mobile/software/solutions/device-management





Mobile Gaming

The Mobile Gaming Economy

Before we start talking about mobile game development we should try to understand what is driving the mobile games market. The rise of mobile gaming since the early days of Java (technically J2ME) games continues to be astounding. Games market research firm, NewZoo recently raised their estimations for the global mobile games market to reach over \$40bn by 2017¹. According to VentureBeat² games make up only 40 percent of all app-store downloads but they represent about 75 percent of spending and the majority of that income is focused on the top 10 grossing games, almost exclusively Free-To-Play. Some of these games have dominated these charts for the last two years. Games like Clash of Clans, Candy Crush and The Simpsons Tapped Out have generated billions and have at least put mobile gaming on the same level as the console market.

It is important to know that for iOS and Android the vast majority of mobile games revenue comes from Free-To-Play games. It is also worth noting that iOS receives about 7% of its revenue from premium games, nearly twice that of Android³.

Making games work on multiple platforms has become easier to do. Around 54% of all mobile games are designed using 3rd party engines and 45% of those use Unity. There are

¹ www.applift.com/blog/mobile-games-market-update.html

² venturebeat.com/2014/11/04/candy-crush-leads-in-u-s-and-u-k-but-clash-of-clans-reigns-in-mobile-crazy-south-korea

³ venturebeat.com/2014/04/25/apple-vs-google-a-world-view-on-the-mobile-gaming-war

many other engines from Cocos, Corona, GameMaker, Unreal, etc. Each engine offers different advantages and perspectives to enable developers of different skill types to quickly realize their ideas and prepare them for release. See the cross-platform chapter of this guide for more information on the available frameworks.

Many developers are under the illusion that they will be the next indie-developer to strike it rich. An activity which happens with games like Flappy Birds gaining crazy numbers of users, but these are closer to 'lottery ticket wins' rather than ideal models to follow. Instead it is important for developers to realize that the mobile games market has become a sophisticated space with many different facets and challenges. Before you start creating your game you need to pay attention to understanding the nature of the market and audience. An essential part of this is that it has become an enormously competitive market, with vast numbers of small teams producing huge volumes of content and spending millions on development and advertising to retain their positions.

Free vs. Paid

Looking in economics terms, it is clear what happens when supply goes up, prices fall. With an effectively infinite supply, the price falls to zero. This is exactly what has happened and why Free2Play is so dominant. But wait, what about the 7% of revenue for iOS on Premium games? Like any market, when dealing with competition, we have a choice. We can either seek volume (create a commodity) or differentiate (create a niche). Successful premium games are the ones which have been able to attract an audience by offering something they perceive to be of greater value than the rest of the games available. Games like Monument Valley or The Room have

shown that this is still possible, and they are noticed because of their premium pricing. This has not been at the scale of the revenues of the top performing Free-To-Play games, despite considerable profiling by app stores.

The movement towards free has not proved an easy path for many developers and attempts to 'clone' the business models of games like Clash of Clans or Candy Crush have rarely seen even a comparable level of success. This is despite the formula appearing on the surface to be so simple, take a simple game mechanic and provide a new social context in order to make it endlessly repeatable. Then add a form of friction which slows the player down from attaining their goals but which is timed to make success always 'just out of reach'. Allow people to pay to remove this friction using consumable goods, but make each goal the trigger to a new goal, also just out of reach.

Of course it is not that easy and this kind of formula is something which can quickly become 'not much fun' and generate a lot of player churn. Even if the developer adds a barrel full of data analytics to find out where players are churning or to work out the best ways to squeeze more cash, in the end the game inevitably dies. Worse than that, the more games feel the same because of the rigid application of a business model over and above the enjoyment of a game, the more players will reject such games. If I feel that the game is merely an exercise in getting me to open my wallet, just how engaged will I be as a player?

What makes these games work is understanding people and building a service which allows them to feel competent, in control and to escape their everyday lives. Revenue comes when we can extend the players delight and engagement over the longer term and give players a reason to want to spend money. We need to build 'lifetime value' not just short term income.

The arguments between Free and Paid games have become almost tribal amongst game developers asking whether business models have tarnished the nature of game design, even asking questions about the morality of these money focused designs. Understanding whether to choose Free or Paid is an essential question for any prospective mobile game developer, but it is not about the business model we would like to follow but the best way to engage the player with the content we are creating to delight them.

The Ethics of Making Money with Games

A lot of the focus of the mobile industry in 2014 has been about the moral and ethical questions behind Paid Apps versus Free-To-Play. Done well, a Free-To-Play game unlocks the value of the free player not just in terms of their viral potential, but in terms of creating a brand and the conditions which encourages the player to pay. However, Free-To-Play games do not have a great reputation, especially amongst parents concerned about in-app purchase and certain game designers. There have been a number of high profile scandals, notably with children 'accidentally' running up \$1000s in-app. I believe that this argument has tarnished the image of the games industry as a whole and got the attention of regulators including the EU who have issued guidelines on the sale of in-app purchases to children. However, to date the regulation seems to have been sympathetic and sensible. Quite rightly, they have asked designers and games retailers to communicate what is for sale and how that is accessed, particularly by an underage audience. It asked the platform holders to make important but minor changes to the way the platforms work (which was largely happening anyway). In short to clarify the expectations which are already enforced by existing legislation.

Another issue raised against Free-To-Play games is that half

of the revenue comes from 0.15% of players, at least according to app testing firm Swrve⁴ only 1.5% of players spend anything on a Free-To-Play game at all. However, this report is problematic not least as it is essentially comparing the 'Purchase' of a Premium Game with the 'Download' of a Free-To-Play game. These are not the same thing. Players will often download a game on a whim, perhaps never playing it. Many games will have different spending profiles. Some games will convert well, others may not convert to spending until later, and each game is different. Even if these extreme numbers were true, would that necessarily be a bad thing? A single player paying \$5,000 for something in a game may seem extreme, but if they gain personal value from doing that is it bad?

The answer to this question depends on a number of factors, but for me comes down to manipulation or addiction. Free-To-Play games designers often talk about Operant Conditioning, and in particular an experiment known as 'The Skinner Box'. These boxes, named after the psychologist who created them, allowed animals to obtain food by pressing a button. The experiment found that varying the rate at which the button released the food affected the behavior of the animals. The premise being that in games designers find ways to give players rewards and this is an equivalent method of conditioning. It is true that some experiments have shown Skinner Box conditioning can work on humans, at least in the short term. In games we do not control all the stimuli or use food (or other low-level needs) as motivation. As designers there are much more powerful methods to retain a player than giving out short term rewards. We can tell stories. We can delight with visual and audible stimulus. We can create games

⁴ recode.net/2014/02/26/a-long-tail-of-whales-half-of-mobile-games-money-comes-from-0-15-percent-of-players

which become shared social experiences. All of these have a much greater effect to stimulate users than any operant conditioning exercise. In the short term we could (but should not) manipulate players, especially vulnerable individuals such as children. Doing so will not last over time and will in the long term reduce our life-time value. It is much more commercially effective to make better games.

Addiction is something we have always used as a shorthand to say that a game was good and compelling. Now it has become a source of concern. Any game will reward players with a dopamine release when they succeed. This creates a physiological response not dissimilar to drug taking or physical exercise. We know in the case of gambling that this kind of body chemistry can create addiction, which as said before is a compulsion that overwhelms otherwise rational behavior. However, gambling has a very different stimulus to games. With gambling the rush is based on the uncertainty and the stake which we put on the line. That uncertainty remains each and every time we gamble. With games we learn the mechanics, reducing the level stimulus over time. Whilst Games Addiction is a recognized problem, it is something that comes under the term of behavioral addiction where an activity becomes compulsive. This is still under research and not currently included in the Diagnostic and Statistical Manual of Mental Disorders, Fifth Edition (DSM-5).

Making The Right Game

Creating delightful experiences for our target audience requires just as much creativity as ever, perhaps more. Players expect us to create joy and to show them new content ideas. Players need something of the familiar in order to be able to compare and help them understand and relate to new content. Scott

Rogers in his book “Level Up” described this as the “Triangle of Weirdness”⁵. He claimed that games consist of a world, activities and characters. We can change any of these for new ideas, but we cannot change all three without risking losing the audience.

For me the kind of fun we are looking for in games is something which happens when the player is able to suspend their disbelief and engage in an experience which has no real-world value. We become totally absorbed in the mechanics and narrative of the experience. Curiously, challenge and frustration are as much the motivations to play as they are potential causes to leave the experience. If we can keep the balance between these states we attain a joyful state that all game designers know about, ‘Csikszentmihalyi’s Flow’⁶.

We have to appreciate that what makes for fun in the mobile space is different from other platforms and may even seem contradictory. We need games which are simple and accessible, but with enough depth and a sense of purpose and progression to retain player attention. If we look at what has been successful and what has not, we see that a mobile game has to give us meaningful success in less than a minute, but keep us playing for hundreds of days. The game has to stop us burning through all the content in a single session, but get us to play dozens of times per day. We need a game which will be familiar, but that will also stand out enough to get featured by app stores. Our game has to be enjoyable (and often free), but still create new reasons for players to want, no, need to spend money with us. The list of apparent contradictions goes on.

We know that there are games developed from emergent mechanics, building blocks which combine to create surprising

5 mrbosdesign.blogspot.co.uk/2008/09/triangle-of-weirdness.html

6 scienceandvalues.wordpress.com/2010/02/26/csikszentmihalyis-flow-pleasure-and-creativity

or strategic outcomes, like Chess or Clash of Clans. Then there are those games built using a series of progressive decision points each resolved with their own steps chained together to make a story such as FTL or Monkey Island. We can even build games which incorporate player creativity such as Createria or Minecraft, or simple abstract puzzles like Threes or SuperHexagon. Whichever path we take, balance is at the heart of our thinking as a designer. We have to decide how much the game will be affected by skill and how much by luck, the extent to which the game follows a fixed narrative or is player led and of course the complexity of the internal systems, whether that is about character development or a resource economy. With Free-To-Play we also have to consider the impact of money spent on the game experience.

What we often forget is to consider what matters to the player. One of the most important questions we should ask is why they should choose to make your game their distraction of choice. Lets face it, Most mobile play starts out as distraction, even if in the end we spend more time on our phones than our consoles. So why would they play your game? Just saying it is a good game is not enough. We have to be able to answer that question honestly. To help us make good decisions we can learn a lot from Classical Design and Product Marketing.

Engaging the Mobile Player

When we develop mobile games we are creating an experience to entertain players on a specific kind of device. Tablets and Phones fulfill different needs and require an attention to detail on the specific and different mode of use they fulfill. The phone is generally about 'the next minute', it is something we get out when we expect something to happen or need something to occupy ourselves. How do players use their tablet devices? For me I think it is about a longer period of rest or relaxation. What does that mean for the game we want to make?

The realization of our game is the combination of a distinctive vision, compelling gameplay narrative and how the experience is designed to affect the emotions of the players. This has to be appropriate to the nature of how the game is consumed and in mobile we have to understand the restrictions inherent to these devices. The limited screen size, Touch screen controls, accelerometers, battery life, the ability to be interrupted, the ease players have to get out and put away the device, the limited speaker quality, the high quality headphone output, etc all affect the way players interact with the device. Mobile phones are (mostly) connected to the internet and are the most pervasive of devices as we always carry them.

To show you what I mean, think about the way we implement controls. Touch screens allow a huge range of movement on a 2D plane, but after a short while our skin will get hot and lose capacitance which means the controls will become less reliable. If we simply try to duplicate a twin stick control system (like too many games have) we will soon find problems with the playing experience. Arguably, this is one of the reasons why First Person Shooters have not been quite as popular on mobile. Instead we should design game mechanics with controls specifically to make the touch

process feel good or which recognize the limits of the methods available to us and make that part of the experience. MiniGore is a great example for me where the difficulty of the game is actively enhanced by the twinstick mechanics becoming harder to control over time. On the other hand, Hayday from Supercell demonstrated how touch could be utterly delightful. The touch motion used to collect your crops is so pleasant that it elevates this game far above other farming games on any platform and was suited for a Tablet experience.

The need for immediate satisfaction of the mobile gamer does not replace longer term engagement. However, it does reinforce the need for simplicity in the gameplay needed for phone based games. Simplicity itself will fail to continue to sustain the level of interest in playing. To keep people playing we need to create a context which gives us a reason to repeat that game mechanic. Something which gives us a sense of purpose and progression. Something which calls for our attention after our playing session has ended and encourages us to return to play. That initial mechanic had better be enjoyable even after thousands of plays.

Games like CSR and Candy Crush introduced this method of game design to the mobile market. Finding ways to chain together a series of grinding mechanics to sustain playability over thousands of sessions whilst always giving a sense that the goals are achievable is magical. It builds long term engagement and keeps players involved with your game ever longer, provided the experience is sufficiently meaningful. Keeping players playing longer has a direct impact in their willingness to spend money in the game. In a 2014 survey by Unity⁷, the reported spend by paying players who spend less

⁷ www.gamesindustry.biz/articles/2014-10-14-mobile-spending-driven-by-35-44-year-olds

than an hour in a game averages at \$0.66, but for those who spend over 10 hours this rises to \$15.15.

Designing the Player's Journey

The realization that long term engagement matters has a profound impact on the way we look at game design. The idea of a game as a mechanic or story is transformed to the realization that it is not only our hero character who is going on a journey, but our player as well.

The first stage of that Journey is Discovery. Players have a particular set of needs and aspirations when they first encounter your game. We have to ensure that the journey to discovery sets up the conditions that will encourage them to download and play the game the first time. There is usually very little opportunity to set the right expectations, but doing so is essential. If the game charges upfront, let the player know why they should still buy it, what they will miss out on if they do not. If the game is free we still have to create expectations, but we also have to show the player why the advertising is worth the hassle or if there are in-app purchases, why those players will feel good about buying them. This is a delicate art. We need to be clear and transparent and still communicate why this game is worth their time and investment.

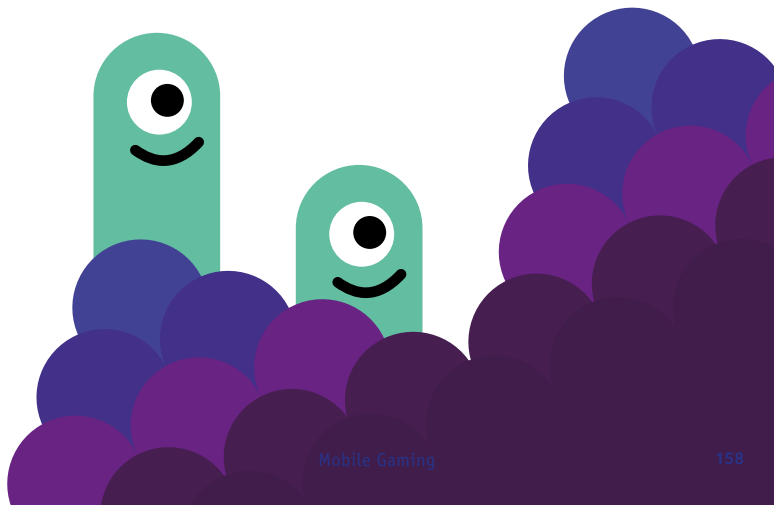
Once they have taken the choice to install your game we have to make it as easy as possible to engage. Make the icon and name of the game instantly recognizable and ideally a tease, a reason to kick off the app. At this stage we do not want them to make choices about what characters to play or what levels they want – they do not know yet. Do not make them sign up to Facebook or set-up an account before playing – show them what the game is all about. Then knock their socks off. I often talk about “The Bond Opening”, comparing this first ever play of the game to the opening 5 minutes of

every Bond movie. It blows us away and at the same time set up everything we need to know about the story, super-secret agents and the world the story takes place in. But it does more than that, it ensures we never want to leave the seat. It is this dual role of showing not telling that sets up the expectations for the rest of the movie which applies to games so well. As this is a game, we don't want to Show or Tell, we need to 'Do!'. Players need to learn about games by doing and feeling good about their achievements quickly.

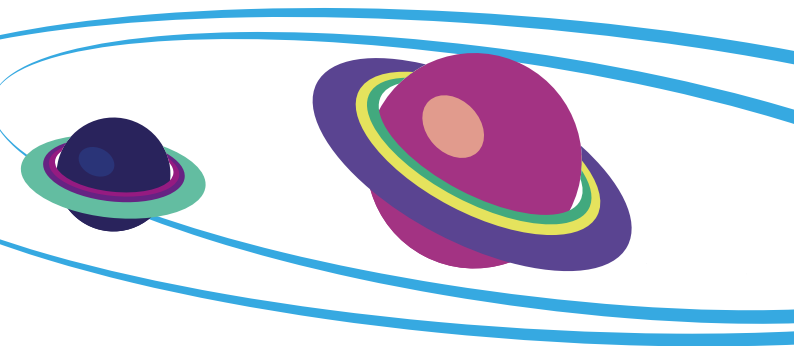
If we succeed and manage to educate the players and set up the right expectations of future value we have to keep them playing. They need reasons to return. For a game to become something we play regularly and for a long time we need to understand what success feels like. We need to know that there will be challenge and progression as well as a sense of purpose. This means that we need to see ourselves fail and still want to continue to play. We need a sense of unfinished business that compels us to overcome the difficulty in playing again. We have to understand the reasons to continue and have a series of achievable goals, all without over-complicating the game. One technique which can help with this is the cliffhanger concept. There are two things which are happening here. First we are accepting that we have to give our audience a natural break from the playing experience and that is important. I am not suggesting we have to prevent players from continuing, but giving them the option to stop is useful for building engagement as long as you give them a reason to return. That reason to return could be to wait for more fuel, for plants to grow, for a vehicle to repair or even for another player to visit your city. The key is to create a sense that if your player does not return they are missing out. Do not punish them. If we can encourage players to want to return regularly and to

create appointments to play then we know that they are truly “engaged”.

Once players are truly engaged, other factors become important if we want to sustain their interest. We do need to move the game forward with their changed needs. This might be in the form of content or other kinds of playing extension. It might also be through Social Play. This means creating commonality, bonds and identification by establishing shared rules. When we are designing games we have to think about how the rules we use to create engagement and entertain also provide the means for people to identify themselves with our game and provide the means for them to express that. Social factors are incredibly important and can have a marked impact, not just in terms of engagement but also in terms of revenue. If buying an in-app purchase will not just help me get more out of the game, but will also make me look good to other players then I am much more likely to do that.



It is interesting that social communities help create a depth of engagement that some have referred to as 'Whales' but I prefer to call "True Fans". These are players who spend significant amounts in your game and who can often be the principle source of your income. In many games these players will not be hugely social, indeed they spend much of the game trying to maximize the effectiveness of play rather than engaging with others. However, the presence of other players creates the conditions which enable those players to emerge. Without the Free Players we tend not to get this True Fan level of engagement. It is important not to confuse these players with people who are addicted. Addiction is where individuals have a compulsion which overwhelms their otherwise rational behavior. In practice the majority of True Fans are rational people who have made your game their principle hobby. Addictive Behavior is always detrimental to the individual and we should do everything we can to help anyone with these kinds of issues.



The final lifestage we have to acknowledge is “Churning”. It is inevitable that in the end players will stop playing our game. We want to delay that as long as possible, but to fail to plan for that is going to cause us more problems. To understand what keeps people playing over an extended time period I get people to think about “The Columbo Twist”. This is based on the classic detective show featuring actor Peter Falk as the eponymous bumbling lieutenant. What made this one of the best television shows of all time is that it did something odd for a murder mystery. You saw who did it. Where is the mystery for a show about murder if you know who did it? Well it turns out that the joy of watching this show was in waiting for Columbo to say those famous four words... “Just One More Thing!” that always happened in the last few minutes of the show. He would have talked to the murderer for the 4th or 5th time talking about the most random seeming evidence. Then he would hit you with that phrase and you knew this was when he would tell you everything about the murder. Not just who did it but why and how Columbo had worked it out. That was the predictable payoff you were waiting for. You knew it was coming and were waiting for it. What in your game keeps your players coming back even when they know everything that is going to happen?

Getting Discovered

If you have followed these guidelines then you will have already put your game design into the best form that suits your audience and that itself will (hopefully) give you a fighting chance. However, that alone is not enough. We have to use every possible communication route we can and that usually requires investment. It is still possible to succeed without spending money on advertising, but you have to be the winner of a global lottery ticket. This applies on mobile games as on any other kind of mobile app as well. Some hints how to market your software can be found in the monetization chapter. Additionally here are strategies which you might want to think about especially for games.

Getting noticed by the press can help, particularly if you participate in Games Awards such as Pocket Gamer's Big Indie Pitch⁸ or the Indie Awards at Casual Connect⁹. If you can get the attention of YouTubers that may also help.

Spending money on advertising can help, but it is important to realize that you are competing with a lot of people and some big players who are seeking large audiences. It is important to remember what you are trying to achieve when creating an advert. There are two motivations, building awareness and direct action (i.e. downloading the game). In games we are able to put adverts in other games and apps on the same device we want the players to experience the game. There is nothing getting in the way between the advert and the app store. One click and you can buy/download the game. That is an amazing thing, no other media has that kind of frictionless experience.

⁸ www.pocketgamer.biz/events

⁹ indieprize.org indieprize.org

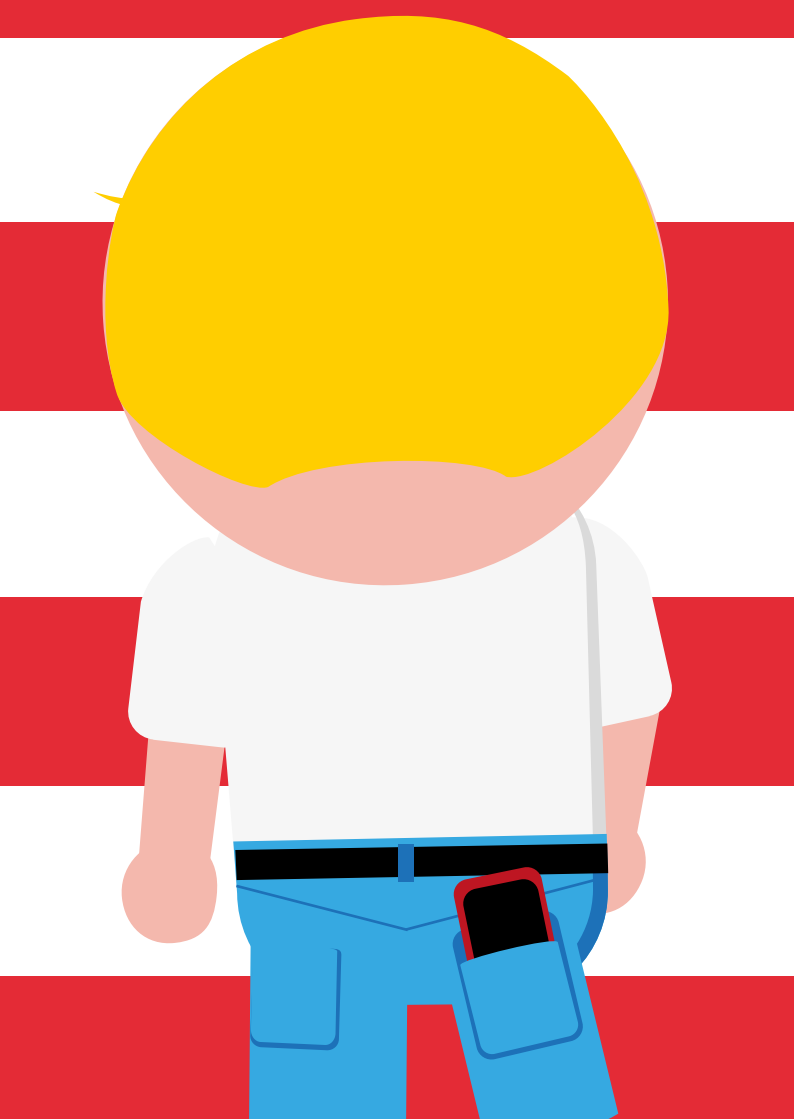
Another peculiarity to be aware of is that the larger the reach (range of players) you are looking for, the more expensive each of the installs. This is because buying space on an advertising network is based on a bidding process and the results will be calculated on the basis of Cost Per Install, Cost Per Mille (i.e. per thousand) or a blend of the two known as eCPM (effective CPM) as well as Ad networks like Chartboost.com or AppFlood.com which offer cross promotion.

Video based advertising is growing and allows the player to instantly understand the nature of the game being shown. This is often combined, such as with Unity Ads¹⁰, Vungle.com and AdColony.com with incentives inside the game – such as free currency. This kind of incentive is different from external incentives such as offered by providers like Tapjoy and importantly is not allowed on Apple's network.

Regular events and outreach to the community allow us to sustain and to grow our audience. Building on genuine social experiences, such as the recording of gameplay videos and sharing of community data (highscores etc) players can help reach out to their friends and other potential players via Facebook, Twitter, Everyplay and Youtube.

In the end despite all the differences in the details, Mobile is like any other platform. We have to acquire, retain and monetize our audience. That only happens if we entertain players in the way that works for their devices. Devices which are perhaps the most social and most pervasive devices in human history. Mobile gaming is thriving despite the hurdles and the lessons learned will affect every aspect of game development.

10 unityads.unity3d.com



Mobile Development & the Internet of Things

Several years ago two giants -the mobile industry and the industrial Internet (also known as m2m, embedded and industrial telemetry)- lived separate lives. Each with trillion dollar turnovers to date, yet with far too little shared knowledge in necessary insights on how to ultimately provide end-to-end services where both giants need to be involved. Some say it is simply due to that mobile companies do not have a strong tradition of working with embedded technology, its formats or protocols bespoke for industrial applications. Others say it is more due to that hardware developers creating services for the Internet of Things (IoT) are unfamiliar or less interested in learning the innards of Android Java programming or the Objective-C language used for conceiving native apps for iOS.

Luckily the tables are starting to turn, where many traditionally closed hardware systems are opening up APIs and even moved codebases to GitHub¹. It looks very promising migrating towards more open standards, and exposing custom interfaces to IP-based technologies. Needs for hooking up phones to connected things is becoming more commonplace, and many hardware devices are found increasingly hampered or even rendered useless without their counterpart app. System functionality is also increased over time after a hardware product is acquired, by upgrading firmware, apps as well as the server-side. The concept of releasing often and early has spread from digital into the physical world, thanks to dynamic software architecture, over-the-air updating not to mention

¹ github.com/

getting users into the habit to use things that hardly work when first purchased.

New Roles for mobile

Early uses for mobile devices were as windows on what your smart devices were doing. These days mobile devices and mobile apps can control IoT devices remotely, and can even act as the sensor itself; where e.g. a user's GPS position is essential for many types of localization of content, and contextual services. In other cases the smartphone is a gateway or proxy to sensors, like in the case of a wearable sports tracker with bluetooth connectivity. Remote controlling, visualization, storage, off-line functionality and well as means of authentication are other probable roles.

Examples of IoT integration with mobile apps:

- **NEST Smart homes:** developer.nest.com
- **Philips Hue LED lighting:** meethue.com
- **The Nike+ product line:** nikeplus.nike.com
- **BackYardBrains' Roboroach**, a remote-control set for cockroaches: backyardbrains.com/products/roboroch

See postscapes.com/internet-of-things-examples for more examples from sectors like health, city infrastructure, environmental monitoring or the classic industry.

Tools of the trade

From a developer's point of view, the occurrence and popularity of the growing range of third-party tools and development kits is a proof point of genuine interest rather than being just a fad. At the time of this writing, mobile IoT is pretty much a two-horse race between iOS and Android SDKs and libraries for

native development. REST APIs, of course, can be made from most mobile apps.

The context may be key

Every market segment of IoT; from wearables to real estate automation, from medical applications to surveillance monitoring has their own special challenges; offline usage, large datasets, need for strong encryption, real-time interaction without delay or high demands on bandwidth. No single tool or library covers them all. The context may limit your implementation options.

What makes an IoT app, you might rightfully ask. It may not seem very different from any old database client app, so what's the big deal? Well, the devil lies in the details, as usual. Let's say you are auditing your power consumption in your house, and there is a gadget connected to your fuse box, it speaks regularly to a service via a RESTful interface over HTTP, and you access a web page riddled with graphs via your tablet. But then your customer wants to read historical data when offline, or read a NFC tag, or perhaps send a friendly SMS reminder when the sauna is the right temperature, or why not scan for some iBeacons over the phone's Bluetooth Low Energy radio. Suddenly your project has snowballed beyond the safety of the web container, not to mention your web projects budget. Then a native solution, or sometimes a hybrid mix of web and native is the way to go.

Going hybrid for rapid prototyping

As a software engineer, there is good reason to look at hybrid tools for development and prototyping, especially when you are on a budget or in a hurry. Web and scripting technologies are intrinsically easier to grok for the novice developer. They also enable faster development: to create a decent UX, and because you can select from various industry-strength third

party libraries in each category to help implement your app. Not all of them relate directly to IoT core technologies, but even within the industrial internet space we see a growing collection of relevant additions to the capabilities of web technologies offered via a mobile browser or a webview component.

One of the more popular base technologies for hybrids is the Apache Cordova project, sibling its commercial incarnation Phonegap². Several commercial hybrid SDKs use Cordova as one cornerstone, noteworthy Worklight (IBM), Salesforce One (by Salesforce), Evthings Studio by Evthings and the Intel XDK. By its open plug-in architecture, web and native components (purposely built for each target OS) can be mixed and matched freely, with the end goal to create either an xCode project for iOS or an Android app for publishing on appstores. There are also tools and libraries available outside of the Cordova family, while few are focused solely on IoT application development, they may well have functionality useful to industrial service development.

Communications and Protocols

One of the standing issues in development for the Internet of Things (IoT) is the occurrence of exotic communication protocols for a mobile programmer, with names like XMPP³, MQTT⁴ and CoAP⁵. Smartphone apps may need ways to communicate using some of these protocols to interact with devices running as IoT. Thankfully some implementations are available such as the Eclipse Paho project which includes an Android client⁶.

² phonegap.com

³ xmpp.org/

⁴ mqtt.org/

⁵ tools.ietf.org/html/rfc7252

⁶ eclipse.org/paho/clients/android/

A web-centric approach may use HTML5 technologies, such as websockets for interoperability. And as many developers have come to realise, many times websockets simply do not cut it for low-overhead communications as most messaging formats are incompatible. To be able to do low-level TCP and UDP based networking, transport security et cetera, other technologies like Chromium sockets (i.e. Berkeley sockets nicely wrapped for javascripters) needed to be introduced. If you do not want to roll your own, the plug-in architecture of tools like Cordova⁷ come in handy. Establishing mobile plug-in support also for TLS (Transport Layer Security) is also a step forward towards end-to-end strong security from sensor to mobile device safeguards IoT services from many of the uncertainties that face web services and APIs exposed to the public Internet.

As a result, a second wave of apps is coming in where IoT apps that converse directly over short-range radio, using low-level IP-based protocols for sensor data and telemetry messaging with a minimum of overhead.

Further Reading

- **Introductory article** comparing protocols used for the Internet-of-Things: electronicdesign.com/embedded/understanding-protocols-behind-internet-things
- **A Cisco view on IoT Application Protocols:** blogs.cisco.com/ioe/beyond-mqtt-a-cisco-view-on-iot-protocols
- **Scaling the Internet of Things** Video by Yodit Stanton recorded at All Your Base Conference 2014: vimeo.com/album/3108317/video/109904567
- **Eclipse IoT protocols:** iot.eclipse.org/protocols.html

⁷ cordova.apache.org

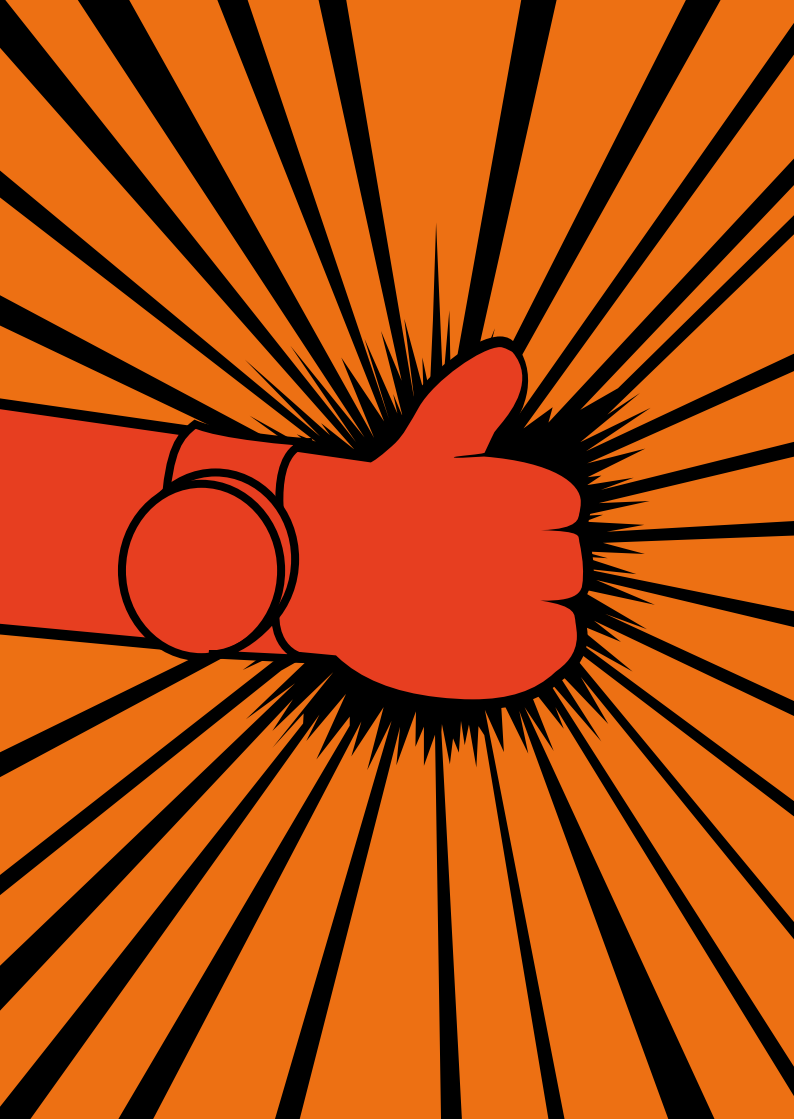
- **Mobile and Web Messaging**, includes MQTT and IoT topics: media.jmesnil.net/slides/2014-11-13_oreilly_webcast.pdf
- **IoT Demonstration using WebSockets**: developer.mbed.org/cookbook/Internet-of-Things-Demonstration

Now go forth and prosper

To end this chapter, here are some good starting points, representing some of the stakeholder of the industry, software, hardware, aggregators and service providers. Here are some of the noteworthy:

- **Appmethod**: appmethod.com/internet-of-things, Cross-platform C++ IoT app tools
- **Estimote**: estimote.com, Manufacturer of iBeacons and mobile SDK
- **Evthings Studio**: www.evthings.com, Rapid IoT prototyping devtools for Android and iOS
- **IFTTT**: ifttt.com, Cloud company connecting events over the internet
- **Intel IoT, and the Intel XDK**: software.intel.com/en-us/iot, Devtools for microcontrollers and mobile apps
- **Parse**: parse.com/products, Back-end company with lots of client code & libraries
- **Phant by Sparkfun**: data.sparkfun.com/, Maker of IoT hardware and accessories, here linking to their nifty IoT server-side backend, perfect for app makers who want to own their own data.
- **Relayr**: relayr.io, IoT hardware + apps company





Programming Smartwatches

After pioneering work of Metawatch, Pebble and many more companies, Google released Android Wear in 2014 and various manufacturers released compatible smartwatches. Samsung released a variety of Tizen powered watches and Microsoft brought its Microsoft Band fitness tracker to the market. Now Apple also unleashes its range of Apple Watches. 2015 might very well be the year of the smartwatch.

The Ecosystems

Arguably the biggest platforms are Pebble, Android Wear, the Apple Watch and Samsung Tizen. There are also Android standalone watches and a whole range of popular activity trackers from companies such as Nike, Jawbone, Fitbit, Misfit, Razer and Microsoft. Most of these devices have a lousy track-record when it comes to battery life, which is why a couple of companies such as Martian, Withings or Cogito fuse traditional time-pieces with smart enhancements. Instead of having runtimes that are measured in days, these watches endure six months or more on a single battery.

An interesting development is that the big smartwatch platforms increase the ecosystem lock in. Android smartwatches require a Google certified Android device that comes with Google Mobile Services, so Android Open Source Platform (AOSP) devices won't do. Samsung Tizen enabled watches work best with Samsung phones and Apple Watch, perhaps unsurprisingly, requires an iOS device to work correctly.

Interaction wise you have to differentiate between standalone apps that run on your watch and companion apps that run on your phone but display content on your watch.

Many smartwatches turn out to be fairly dumb when removing the connected phone, but with some you can even add a SIM card and make calls using the watch directly.

It will be interesting to see how consumers outside of the gadget crazy wave of first adopters will react to these new options. While I am pretty sure that this sector will have to evolve more to gain mass adoption, I also think that smartwatches are here to stay. My prediction for the next version of this guide: we will integrate the watch specific parts to the relevant platform chapters. So now that you support phones and tablets and possibly PCs, please go ahead and add another form factor to your development plans – and while you are at it, do not forget TVs and cars!

Designing UX for Smartwatches

Whatever platform you might chose: Pay attention to the user experience of your smartwatch apps. As you have very little space you need to bring your point across very clearly and without superfluous information. Do not bug your users with too many notifications or by requiring precise input. some devices support touch interactions, but others use traditional watch buttons only. Touches and other gestures may be hard to do correctly on a tiny watch face especially when the user is on the move.

Consider which notifications would be useful to the user, and whether it's practical to allow the user to pick their preferred notification, for instance vibration when in a meeting, and so on.

- Pebble provides an excellent UX guide as part of their design best practices¹. The guide includes Navigation, Design Patterns, and patterns of interaction.
- An in-depth article is available from Nielsen Norman Group which combines a review of the Samsung Galaxy Gear smartwatch with design guidelines for smartphone apps².
- Jonathan Kohl wrote a comprehensive article on designing products for smartwatches and wearables³.

Android Wear

Android Wear targets – as the name suggests – more than smartwatches, but as of writing this chapter the only Android Wear compatible devices out there are smartwatches. In 2014 vendors such as LG, Motorola, Asus, Sony and even Samsung released Android Wear powered watches.

Your starting point for development is developer.android.com/wear. You will always need an Android app that contains the wearable app. You can choose different integration levels for supporting smartwatches:

- No integration: your notifications will still be shown on a connected smartwatch. Bear in mind that this sometimes lead to a notification overload for the user, so be sparse with notifications.
- Android Wear enhanced notifications: you can optimize notifications for display and interaction on the smartwatch.

¹ developer.getpebble.com/guides/best-practices/design

² nngroup.com/articles/smartwatch

³ kohl.ca/2014/lessons-learned-when-designing-products-for-smartwatches-wearables

You can add pages, big views and smartwatch specific actions to your notifications.

- Voice action: you can register voice actions that are triggered on the watch to allow a hands free interaction with your app.
- Wearable app: you can create apps that run on the smartwatch directly and thus have access to sensors etc. This is for example useful for an activity tracker app that allows to track routes without needing to carry your phone with you at the same time (this requires a GPS enabled watch, of course). You can use most Android APIs in your wearable app, only few libraries are not supported: the packages `android.webkit`, `android.print`, `android.appwidget`, `android.app.backup` and `android.hardware.usb` cannot be used.

You can use the Android Wear emulators for testing purposes, but a real device allows you to finetune the experience better. For general Android development please refer to the Android chapter. Keep up with the latest Android Wear developments by joining the Android Wear Developers community⁴.

Apple Watch

The first range of Apple Watches will be released early 2015. Apple Watches come in two sizes and a variety of color schemes to cater for different tastes (and pocket depths).

Start development by visiting developer.apple.com/watchkit. While you cannot create pure standalone apps with the initial version of the WatchKit, you can use these options:

⁴ plus.google.com/communities/113381227473021565406

- **Actionable notifications:** create notifications that are displayed on the Apple Watch and allow the user to interact with them.
- **Glances:** A read-only rich information.
- **WatchKit Apps:** apps can contain WatchKit extensions that run in the background of your iPhone and remotely update and interact with the UI that is displayed on the AppleWatch.

Samsung Tizen

Samsung's initial range of smartwatches operated on proprietary Android forks. In 2014 Samsung started to offer dedicated Tizen-based “Gear” watches and even rewrote the firmware of their existing watches to use Tizen. Currently the standalone-capable, curved Samsung Gear S is surely one of the most iconic smartwatches out there.

Your starting points for Tizen smartwatch development are developer.samsung.com/samsung-gear and developer.tizen.org. You can start supporting Tizen smartwatches by sending rich notifications⁵ that are actionable . The easiest way to develop Tizen standalone smartwatch apps is to embed a Tizen HTML5 app within your Android app. For communicating between your phone based app and your Tizen app you have to use the SAP SP (Samsung Accessory Protocol Service Profile, a name only a mother can love) – basically a byte-array based protocol that requires your own serialization. For general Tizen development please refer to the respective chapter.

To keep up with the latest Samsung Gear news follow the Samsung development Twitter channel [@samsung_dev](https://twitter.com/samsung_dev).

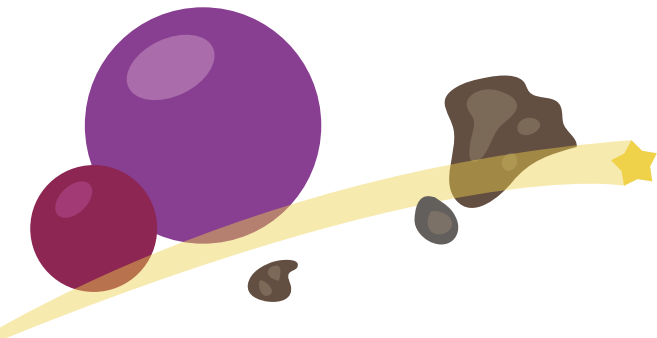
⁵ developer.samsung.com/galaxy#rich-notification

Pebble

Pebble is with Metawatch one of the pioneers of the modern smartwatch movement. As of writing there are two distinct editions of the Pebble watch: the original and the steel pebble. Hardware-wise they both feature the non-touch-enabled-but-power-efficient e-Paper-display with a resolution of 144x168 pixels and 24 kb RAM for apps.

Your starting point for pebble development is developer.getpebble.com. Standalone apps are written in C. You can either use the browser based cloudpebble IDE⁶ on any OS or the Pebble SDK on Mac and Linux systems. You can also use Javascript for developing companion apps that are executed on the phone but can display status updates and more on the phone. An early unofficial version of an emulator is available at GitHub⁷. With background apps, access to sensors and AppMessage/AppSync communication options you can create great Pebble apps.

Follow Pebble on Twitter via [@PebbleDev](https://twitter.com/PebbleDev).



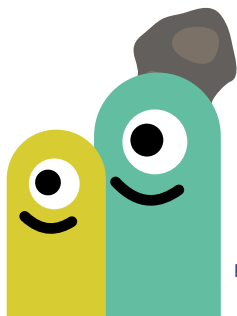
⁶ cloudpebble.net

⁷ github.com/PebbleDev/qemu_pebble

Activity Trackers

There are many activity trackers with associated developer opportunities. Often the only option is to get access to cloud data so that you can create your own statistics, but some devices also support standalone apps. These are the most popular trackers along with the corresponding developer site:

- **Fitbit**, dev.fitbit.com
- **Garmin**, developer.garmin.com/connect-iq
- **Jawbone**, jawbone.com/up/developer
- **Misfit**, build.misfit.com
- **Microsoft Band**, microsoft.com/microsoft-band (SDK announced)
- **Nike**, dev.nike.com
- **Polar**, developer.polar.com
- **Razer Nabu**, developer.razerzone.com/nabu





Mobile Analytics

Mobile Analytics is unlikely to be the first thing we consider when developing mobile apps. Yet it is a great way to understand your app in-the-field, when real users are using our app. App stores provide some headline data and even information about crashes in our app. We can learn much more by designing and implementing Mobile Analytics and we will cover the essential aspects in this chapter.

Data from Mobile Analytics can help many aspects of our work, including the business aspects and operations. We can also learn ways to improve the ways we develop and test the software. In all the excitement we need to remember to protect user's privacy and respect their preferences and expectations. The effects of Mobile Analytics can upset users by consuming valuable resources, or abusing sensitive information about the user and their use of the app.

Why bother?

Mobile Analytics can add value to your organisation and your app, and even to the users of your app if you use the information wisely to improve what you do. The data captured can be used to target your work and reduce inefficiencies. You would be in good company: Over half of the top mobile apps already include mobile analytics¹.

There is an incredible richness in the mobile galaxy where your software can be used on many alien devices that exhibit significant differences in performances and behaviours.

¹ blog.velt.com/mobclix-index-the-when-where-what-of-apps, static.usenix.org/event/sec11/tech/slides/enck.pdf

Researchers discovered battery drain varied by 3x when their app was used on devices with similar hardware specifications, and they even ended up adding custom code to reduce the screen's brightness while their app ran on Kindle Fire's to improve battery life by 40%. They also discovered users who used a keyboard with their app used their app for significantly longer.

Mobile DevOps

Mobile DevOps enables developers to get closer to the operational aspects of their software. App Stores complicate DevOps as the release process is beyond the direct control of the developers and some aspects of data collection is managed and retained by the App Store provider. Mobile Analytics can help collect equivalent information and more application-specific data on how your app is being used. This data can help improve DevOp aspects of the app by providing timely and relevant feedback to help improve the operation of the app. You can also use the data to help you improve future releases of the app.

Getting Started

At least 20 companies offer a smorgasbord of mobile analytics solutions with multiple flavours ranging from campaign tracking to improving software quality. Many include extra features such as crash reporting, customer and revenue tracking. Nearly half offer opensource implementations of their libraries, possibly to allay fears of how their libraries behave?²

² readwrite.com/2013/12/05/why-mobile-developers-need-open-source-analytics-embedded-in-their-applications

Many providers of mobile analytics solutions offer a 'Getting Started' section where you learn how to take your first steps with their products. Examples include Flurry³ and KISSmetrics⁴. You often need to register before you can use the products, as many need configuring with a unique 'key' for your app.

Consider several of the potential solutions before committing to any of them. Read documentation and example code to see how easily you can implement them into your app, and check the legal agreements, including privacy. Then pick at least one of them so you can experiment with implementing mobile analytics into your app. By integrating their code, you are likely to learn much more about what you would like to achieve by using mobile analytics in your app, and how mobile analytics works in practice. Finally read about what other apps use and why they chose them, for instance VentureBeat found 95% of Android developers use Google Analytics, yet "Despite Google's massive market share, fewer than a third of mobile developers consider it their primary app analytics solution."⁵

For multi-platform apps you may want consistency across each platform. Otherwise you may be trying to compare dissimilar, or even disparate, data sets - particularly if different mobile analytics solutions are used for the various platforms. Consider picking a common solution that supports every platform you want to launch your app on.

Two providers are well worth studying. Segment.io⁶ abstracts a wide range of other mobile analytics offerings and they provide their code as opensource at

³ support.flurry.com

⁴ support.kissmetrics.com/getting-started/overview

⁵ venturebeat.com/2014/12/02/230-developers-and-1-8m-apps-reveal-the-best-mobile-app-analytics-solutions/

⁶ segment.io

github.com/segmentio. They demonstrate ways to implement tracking in ways that reduce the effort needed to adapt to different analytics providers. Count.ly⁷ provide opensource implementations of their server as well as of their client libraries and they encourage you to create a complete test environment to evaluate their product.

Deciding What To Measure

What would you like to measure to understand how the app is being used? Some suggestions are:

- **Key usage events:** For instance, new search option or when users launch social networking from your app.
- **Business-centric events:** Any interaction by the user that generates revenue for you. How often do your users purchase the premium version of the app or other items offered within your software? When do they cancel orders or discard their shopping cart before checking out?
- **Application-centric events:** Performance, Usability, Reliability, and other data about the behaviour of the app.

Once you have defined your main areas of interest, you will need to design the analytics measures, for instance, what data elements need to be reported.

Defining How To Measure

Create meaningful names for your interaction events so you can easily and correctly remember what they measure. For each event you want to record, decide what elements it needs to

⁷ count.ly

include. Consider how the data will be used once it has been gathered, for instance sketch out typical reports and graphs and map how the various data elements will be processed to generate each report and graph.

Also remember to address globalization issues such as the timestamp of each element. Does the app detect the time of an event according to the device's location, the device's settings or does it use a global time like UTC time⁸?

Some of the Mobile Analytics solutions will automatically record and report data elements to the server. It is worth checking what these elements are, how and when they are reported, and how they are formatted. Then you can decide whether you want to use and rely on these automatically-reported elements.

Custom event tags augment predefined events, and many mobile analytics solutions provide ways for your app to generate them. You may need to format the custom event messages. If so, pay attention to encoding of the elements and separators. For instance they may need to be URL encoded⁹ when they are sent as REST messages¹⁰.

You may want to consider how often the app should report events to reduce the risk of flooding the available capacity of the analytics system, which might affect the reliability and accuracy of the delivered analytics data. Localytics has some good integration tips online¹¹. One method to reduce the volume of data processed by the analytics solutions is called sampling. Adam Cassar published an interesting blog post on this topic¹².

8 en.wikipedia.org/wiki/Coordinated_Universal_Time

9 en.wikipedia.org/wiki/Percent-encoding

10 msdn.microsoft.com/en-us/library/live/hh243648

11 support.localytics.com/Integration_Overview

12 periscopix.co.uk/blog/should-you-be-worried-about-sampling

Adjusting Your Code

You may need to declare additional capabilities required in order for the mobile analytics to function correctly when integrated with your app.

For Android these are known as permissions. The analytics probably need Internet permissions so the events can be re-reported online, and location-centric permissions if the solution records the location of the phone. If your app already uses the permissions, you do not need to specify their use again.

For iOS, `UIRequiredDeviceCapabilities` tells iTunes and the App Store what device-related features the app needs. It is implemented as a dictionary where the elements are specified using keys. Keys include wifi, location-services and gps.

For Windows Phone 7 and 8.0, capabilities are used to decide what the app uses. Localytics has a quickstart guide online¹³ that includes an example of setting the `ID_CAP_IDENTITY_DEVICE` capability. Windows Phone 8.1 recommends App Specific Hardware ID (ASHWID) instead¹⁴.

Handling the results

There is a lag from when an app sends an analytics event to when the information is processed and made available to you. The lag, or latency, varies from near 'real-time' to many hours. You, and your business sponsors, need to decide how long you can afford to lag real-time events.

Some analytics solutions provide an API to allow you to access the data. This may give you more greater scope to create

¹³ localytics.com/docs/windows-phone-7-integration/

¹⁴ msdn.microsoft.com/en-us/library/windows/apps/jj553431.aspx

custom reports. Several allow you to host the servers which provides you greater control of the data and how it is used.

To evaluate the quality of the results some organizations invest the extra effort of incorporating several analytics solutions into their app and cross-reference the results. However, two conflicting results do not make reconciliation easy, so it may be necessary to use three sets of results to diagnose the differences by triangulation¹⁵.

If you have decided to work with KISSmetrics, check out their article on ways to test your metrics¹⁶.

What can go wrong?

The road to hell is paved with good intentions, there are many things that can go wrong when implementing analytics in mobile apps. Some of the most common include:

- **Uncalibrated results:** blindly trusting the data can lead to a maelstrom of problems. The result can be inaccurate and misleading which causes knock-on problems when you use these results to manage the business and your work. Good practice is to test the analytics implementation at the outset, starting with no users, then one, before testing with more users. Look at latency, accuracy, and reliability of the recorded data.
- **Betraying trust:** Users implicitly trust apps to behave nicely on their mobile devices. However apps may accidentally or deliberately break that trust, for instance by tracking users, recording and then using sensitive data etc. Try not to hide behind click-through agree-

¹⁵ [en.wikipedia.org/wiki/Triangulation_\(social_science\)](https://en.wikipedia.org/wiki/Triangulation_(social_science))

¹⁶ support.kissmetrics.com/getting-started/testing-km

ments which we knew few people read and even fewer understand. Instead, make sure your app and any analytics libraries you use behave nicely and "Do as you would be done by and don't snoop."

- **Handing over the jewels:** Make sure that you do have sufficient rights and access to the data that is collected by the Analytics software. This is especially relevant when using third-party libraries and services.

Be aware, some mobile analytics solution providers may use data reported by your app and they may provide and sell it to others. They may control the life of that data, which means they could make it inaccessible to you. Conversely they may preserve and use it long after you have retired your app. If there is personally identifiable information in the data, there may be additional legal and privacy implications. So it is worth considering how third-parties will use and share the data reported via their software and APIs.

Privacy

Your apps are used remotely by people you may never meet. Apps can be silent observers of users and their use of that app.

Remember to explain to the end-users that the app is designed to record and share information about how the app is being used, ideally in your terms and conditions. You may need or want to enable users to decide if they want their use of the app to be tracked. If so, make it easy for the user to control the settings; and consider providing the user a way to access the recorded data, delete it, or contact the analytics solution provider.

Providers of third-party libraries seem to have a range of attitudes to privacy. Some claim the privacy of users is paramount and stresses the importance of not tracking users. Google Analytics clearly prohibit tracking personally identifiable information in their terms of service¹⁷. Others provide examples, including snippets of source code, that demonstrates how to record clearly personally identifiable data. For instance, KISSmetrics provides the following code snippet¹⁸:

```
[[KISSMetricsAPI sharedAPI]
identify:@"name@email.com"]];
```

And mixpanel provides an example of how to update a user's People Analytics record¹⁹

There are several places to learn more about privacy and ethics of working with data related to users, e.g.:

- Jeff Northrop's blog post on Mobile Analytics²⁰
- Kord Davis' book "Ethics of Big Data" (O'Reilly, 2012)²¹

¹⁷ google.com/analytics/terms/us.html

¹⁸ support.kissmetrics.com/apis/objective-c

¹⁹ mixpanel.com/docs/people-analytics/android

²⁰ jnorthrop.me/2012/07/2/privacy-considerations-mixpanel-people-analytics

²¹ available at shop.oreilly.com/product/0636920021872.do

Learn More

We hope this chapter has whetted your appetite to learn more about Mobile Analytics. Here are some places to start your ongoing research:

- Capturing Mobile Experience in the Wild: A Tale of Two Apps²², a study from the University of Wisconsin highlighting the importance of application-centric analytics based data collected on 1M+ users over 3 years.
- The Beginner's Guide To App Analytics²³, available as a free download.
- The Mobile Developer's Guide to the Parallel Universe²⁴, a sister book to this one, covers Mobile Analytics from a marketing perspective.
- TNW provides a good introduction article²⁵ into the topic from a developer's perspective.
- Upsight²⁶ provides a wide range of whitepapers, on-demand webinars and other materials on mobile analytics.

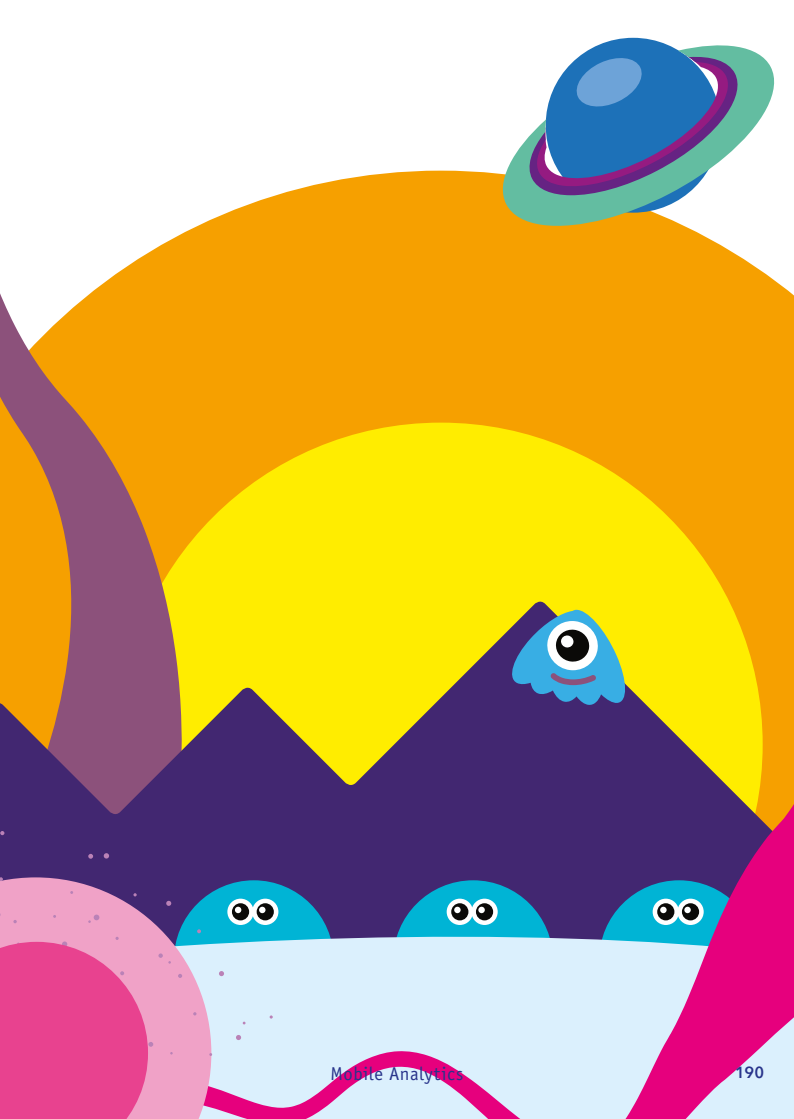
²² static.googleusercontent.com/media/research.google.com/en//pubs/archive/41590.pdf

²³ info.localytics.com/download-beginners-guide-to-app-analytics

²⁴ wip.org/resources/#mobile-developers-guide-parallel-universe

²⁵ thenextweb.com/dd/2013/08/11/9-tools-to-help-you-measure-mobile-analytics

²⁶ upsight.com/resources



APPLICATION SECURITY

Application Security

Readers of this guide know how widespread smart mobile devices have become and how useful mobile apps can be. Mobile devices are also much more personal than personal computers ever have been. People wake up with their phones, stay close to them all day, and sleep next to them at night. Over time they become our trusted ‘partners’.

Many of these apps take advantage of this closeness and trust. For instance, your phone might be treated as part of the authentication for accessing your bank account. Or your tablet could get direct access to the online movies you have bought. The device might even store a wallet of real money for making payments with Near Field Communications (NFC), or virtual money like Bitcoins.

Mobile apps are attracting the attention of hackers and thieves whose interests extend well beyond getting a 99 cent app for free. In Q3 2014 Kaspersky Lab detected 74,489 new malicious mobile programs¹. The historical network and endpoint based defenses (like anti-virus tools) are not enough. Embedding security into the mobile application is critical.

The architecture of mobile apps continues to evolve. Some apps are native-only, and require distinctly different code bases for each different mobile operating system. Some are web-views, little more than a web site url wrapped in an icon. Others are hybrids, a combination of native app functionality with web views. Most mobile apps need to connect with backend services using web technologies to fetch or update information. Like web apps, classic application security needs to be used with mobile apps. Input needs to be validated for

1 securelist.com/analysis/quarterly-malware-reports/67637/it-threat-evolution-q3-2014

size, type, and values allowed. Error handling needs to provide useful error messages that do not leak sensitive information. Penetration testing of applications is needed to assure that identification, authentication and authorization controls cannot be bypassed. Storage on the devices needs to be inspected and tested to assure that sensitive data and encryption keys are not stored in plain text. Log files must not capture passwords or other sensitive information. SSL configurations need to be tested.

Users want to use your applications safely; they do not want unwelcome surprises. Their mobile phone may expose them to increased vulnerabilities, for instance potentially their location could be tracked using an inbuilt GPS. The camera and microphone could be used to capture information they prefer to keep private, and so on. Applications can also be written to access sensitive information such as their contacts. And malicious applications can covertly make phone calls and send SMS messages to expensive numbers.

The application developer may be concerned about his/ her reputation, loss of revenue, and loss of intellectual property. Corporations want to protect business data which users may access from their mobile device, possibly using your application. Can their data be kept separate and secure from whatever else the user has installed?

Threats to Your Applications

On some platforms (iOS and Android in particular), disabling the built-in signature checks is a fairly common practice. You need to consider whether or not it would matter to you if someone could modify your code and run it on a jail-broken or rooted device. An obvious concern would be the removal of a license check, which could lead to your app being stolen

and used for free. A less obvious, but more serious, threat is the insertion of malicious code (malware) that could steal your users' data, or inject illicit content and destroy your brand's reputation.

Reverse-engineering your app can give a hacker access to a lot of sensitive data, such as the cryptographic keys for DRM-protected movies, the secret protocol for talking to your online game server, or the way to access credits stored on the phone for your mobile payment system. It only takes one hacker and one jail-broken phone to exploit any of these threats.

If your application handles real money or valuable content you need to take every feasible step to protect it from Man-At-The-End (MATE) attacks. And if you are implementing a DRM standard you will have to follow robustness rules that make self-protection mandatory.

Protecting Your Application

Hiding the Map of Your Code

Some mobile platforms are programmed using managed code (Java or .NET), comprised of byte code executed by a virtual machine rather than directly on the CPU. The binary formats for these platforms include metadata that lays out the class hierarchy and gives the name and type of every class, variable, method and parameter. Metadata helps the virtual machine to implement some of the language features (e.g. reflection). However, metadata is also very helpful to a hacker trying to reverse engineer the code. Decompiler programs, freely available, regenerate the source code from the byte code, and make reverse engineering easy.

The Android platform has the option of using the Java Native Interface (JNI) to access functions written in C and compiled as native code. Native code is much more difficult to

reverse engineer than Java and is recommended for any part of the application where security is of prime importance.

“gcc” is the compiler normally used to build native code for Android, its twin-sister “clang” is used for iOS. The default setting for these compilers is to prepare every function to be exported from a shared object, and add it to the dynamic symbol table in the binary. The dynamic symbol table is different from the symbol table used for debugging and is much harder to strip after compilation. Dumping the dynamic symbols can give a hacker a very helpful index of every function in the native code. Using the `-f visibility` compiler switch² correctly is an easy way to make it harder to understand the code.

Compiled Objective-C code contains machine code and a lot of metadata which can provide an attacker with information about names and the call structure of the application. Currently, there are tools and scripts to read this metadata and guide hackers, but there are no tools to hide it. The most common way to build a GUI for iOS is by using Objective-C, but the most secure approach is to minimize its use and switch to plain C or C++ for everything beyond the GUI.

Hiding Control-Flow

Even if all the names are hidden, a good hacker can still figure out how the software works. Commercial managed-code protection tools are able to deliberately obfuscate the path through the code by re-coding operations and breaking up blocks of instructions, which makes de-compilation much more difficult. With a good protection tool in place, an attempt to de-compile a protected binary will end in either a crashed de-compiler or invalid source code.

De-compiling native code is more difficult but can still be

² <http://gcc.gnu.org/wiki/Visibility>

done. Even without a tool, it does not take much practice to be able to follow the control-flow in the assembler code generated by a compiler. Applications with a strong security requirement will need an obfuscation tool for the native code as well as the managed code.

Protecting Network Communications

Network communications are also vulnerable, particularly when apps can be installed in emulators or simulators, where network analyzers are freely available and able to monitor and intercept network traffic. Protect all proprietary communications using HTTPS. Even then MATE attacks, especially over WiFi networks, may disclose sensitive data. One way to step up transport security is to use asymmetric encryption between the server and the mobile app (using public/private key pairs) to provide end-to-end security. For sensitive corporate data and applications, install Virtual Private Network (VPN) servers, and install VPN clients on the mobile devices. VPNs generally provide strong authentication, and secure transport above and beyond HTTPS.

Protect Against Tampering

You can protect the code base further by actively detecting attempts to tamper with the application and respond to those attacks. Cryptography code should always use standard, relatively secure cipher algorithms (e.g. AES, RSA, ECC), but what happens if an attacker can find the encryption keys in your binary or in memory at runtime? That might result in the attacker unlocking the door to something valuable. Even if you use public key cryptography and only half of the key-pair is exposed, you still need to consider what would happen if an attacker swapped that key for one where he already knew the other half. You need a technique to detect when your code

has been tampered. Tools are available that encrypt/decrypt code on the fly, run checksums against the code to detect tampering, and react when the code has changed.

Communications can be monitored and hacked between the mobile app and backend services. Even when using HTTPS, an intercepting web proxy (like Paros) can be setup on a WiFi connection that will inspect encrypted traffic. Attackers can then tamper with the data in transit, for profit or fun. So if really sensitive data is being sent via HTTPS, consider encrypting/decrypting application data between the mobile application and the server, so that network sniffers will only ever see encrypted data.

Protecting Cryptographic Algorithms

An active anti-tampering tool can help detect or prevent some attacks on crypto keys, but it will not allow the keys to remain hidden permanently. White-box cryptography aims to implement the standard cipher algorithms in a way that allows the keys to remain hidden. Some versions of white-box cryptography use complex mathematical approaches to obtain the same numerical results in a way that is difficult to reverse engineer. Others embed keys into look-up tables and state machines that are difficult to reverse engineer. White-box cryptography will definitely be needed if you are going to write DRM code or need highly-secure data storage.

Best Practices

Do Not Store Secrets or Private Info

Minimize the amount of sensitive information stored on the device. Do not store credentials or encryption keys, unless secure storage is used protected by a complex password.

Instead, store authentication tokens that have limited lifetime and functionality.

Log files are useful for diagnosing system errors and tracking the use of applications. But be careful not to violate the privacy of users by storing location information, or logging personally identifiable information of the users. Some countries have laws restricting the tracking information that can be collected -- so be sure to check the laws in the countries in which your app will be used.

Do not print stack traces or system diagnostics that hackers can leverage to penetrate further.

Do Not Trust The Device

When you design an application, assume that the device will be owned by an attacker trying to abuse the app. Perform the same secure software development life cycle when building mobile apps as you would for backend services. Do not trust even the databases you create for your mobile apps -- a hacker may change the schema. Do not trust the operating system to provide protection -- many OS protections can be bypassed trivially by jailbreaking the device. Do not trust that native keystores will keep data secret -- some keystores can be broken by bruteforce guessing unless the user protects the device with a long complex password.

Minimize Permissions

Android has the concept of permissions, iOS has entitlements, which allow the application access to sensors such as the GPS and to sensitive content. On Android these permissions need to be specified as part of creating the application in the AndroidManifest.xml file. They are presented to the user when they choose to install the application on their device.

Each permission increases the potential for your application

to do nefarious things and may scare off some users from even downloading your application. So aim to minimize the number of permissions or features your application needs.

Tools

Protection

Basic Java code renaming can be done using Proguard³, an open-source tool and Arxan's GuardIT⁴.

Two vendors for managed-code (Java and .NET) protection tools are Arxan Technologies⁵ and PreEmptive Solutions⁶.

The main vendors for native code protection tools and white-box cryptography libraries are Arxan and Irdeto⁷.

Techniques for protecting Android code against tampering are documented at androidcracking.blogspot.com/. Arxan's EnsureIT allows you to insert extra code at build time that will detect debuggers, use checksums to spot changes to the code in memory and allow code to be decrypted or repaired on-the-fly.

Sniffing

A standard free web proxy tool is Paros⁸. A standard network sniffing tool available on common platform is Wireshark⁹.

³ www.proguard.sourceforge.net

⁴ arxan.com

⁵ arxan.com

⁶ preemptive.com

⁷ www.irdeto.com

⁸ sourceforge.net/projects/paros

⁹ sourceforge.net/projects/wireshark

De-Compiling

See the Hex Rays de-compiler¹⁰.

Learn More

Here are some useful resources and references which may help you:

- Apple provides a general guide to software security¹¹. It also includes several links to more detailed topics for their platform.
- Commercial training courses are available for iOS and Android. Lancelot Institute¹² provides secure coding courses covering iOS and Android.
- O'Reilly published Jeff Six's book on Android security called Application Security For The Android Platform¹³ and another for iOS by Jonathan Zdziarski: Hacking and Securing iOS Applications¹⁴.
- Charlie Miller et al. published the iOS Hackers Handbook¹⁵,
- which demonstrates how easy it is to steal code and data from iOS devices.
- Academic researchers demonstrate how much information can be gleaned from public Android apps at USENIX 2011¹⁶.

¹⁰ www.hex-rays.com

¹¹ developer.apple.com/library/mac/navigation/#section=Topics&topic=Security

¹² www.lancelotinstitute.com

¹³ shop.oreilly.com/product/0636920022596.do

¹⁴ shop.oreilly.com/product/0636920023234.do

¹⁵ www.wiley.com/WileyCDA/WileyTitle/productCd-1118204123.html

¹⁶ static.usenix.org/event/sec11/tech/slides/enck.pdf

- A free SSL tester is provided by Qualys Labs¹⁷.
- Extensive free application security guidance and testing tools are provided by OWASP¹⁸, including the OWASP Mobile Security Project¹⁹.
- An open-source mobile application performance monitoring tool for Android is provided by AT&T's Application Resource Optimization tool²⁰.

The Bottom Line

Mobile apps are becoming ever more trusted, but they are exposed to many who would like to take advantage of that trust. The appropriate level of application security is something that needs to be considered for every app. In the end, your app will be in-the-wild on its own and will need to defend itself against hackers and other malicious threats, wherever it goes.

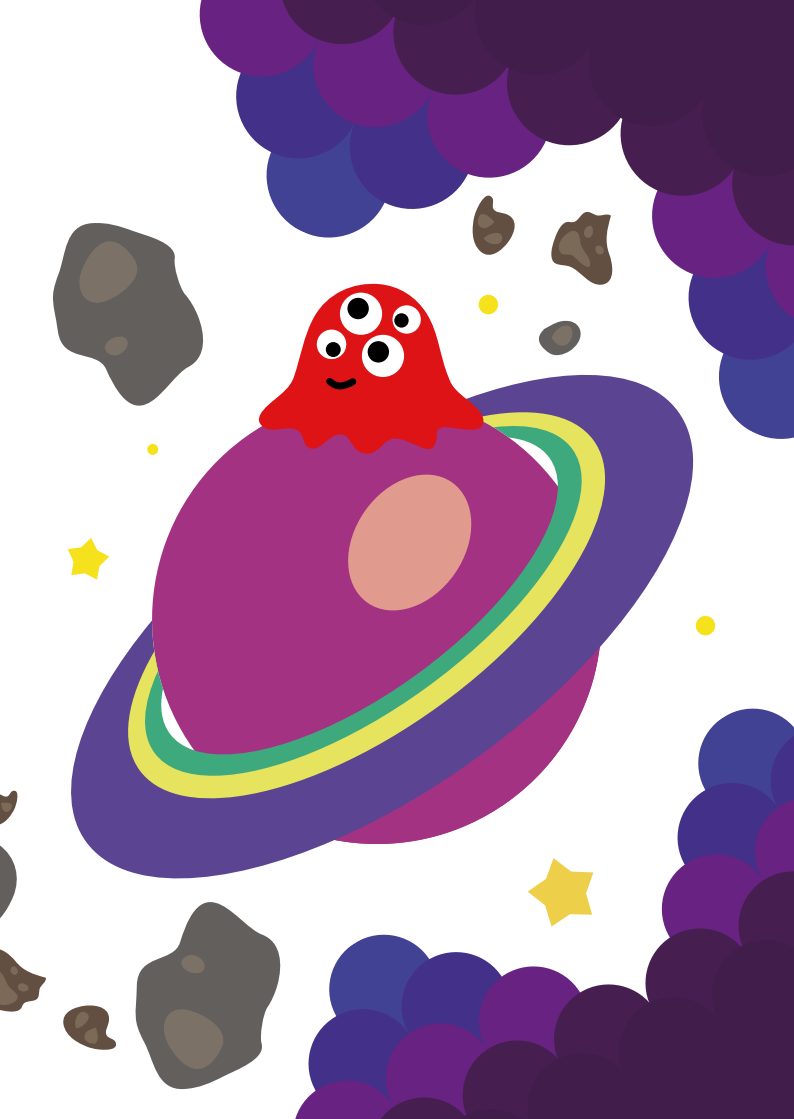
Invest the time to learn about the security features and capabilities of the mobile platforms you want to target. Use techniques such as threat modelling to identify potential threats relevant to your application. Perform code reviews and strip out non-essential logging and debugging methods. Consider how a hacker would analyze your code, then use similar techniques, in a safe and secure environment, against your application to discover vulnerabilities and mitigate these vulnerabilities before releasing your application.

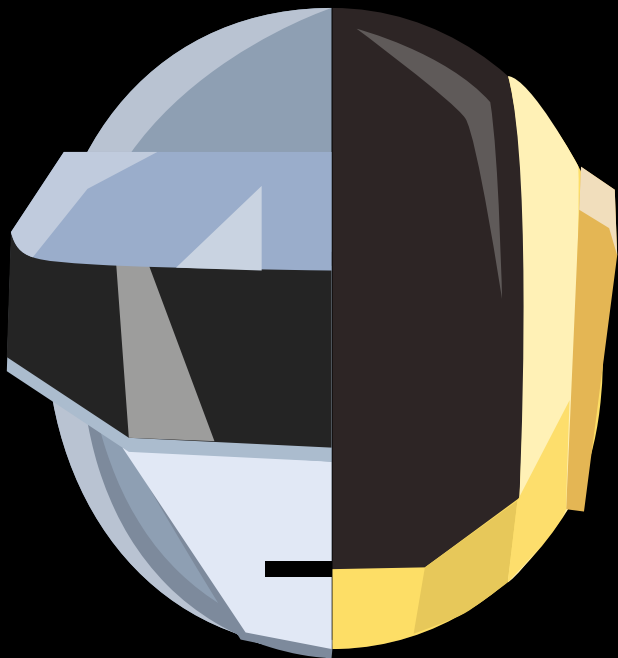
¹⁷ www.ssllabs.com/ssltest

¹⁸ www.owasp.org

¹⁹ www.owasp.org/index.php/OWASP_Mobile_Security_Project

²⁰ decom/application-resource-optimizer





Accessibility

In December 2014 the World Health Organization (WHO) stated on their website that over 15% of the world's population have some form of disability¹ and rates of disability are increasing due to population ageing and increases in chronic health conditions, among other causes. This means that around 1 billion potential users could have difficulties using your app if your app is not accessible.

There has been a huge increase in smartphone and tablet use in the general population, this is no different for those with disabilities. This is because they can buy a product off the shelf that has accessibility built-into the OS when traditionally they would have had to buy expensive add-ons. The WebAIM Screen Reader Survey² shows that there has been an astounding increase in smartphone use by blind people who use screen readers. Older people might not have used a computer at work; however they are finding that they can get to grips with touch screen devices more quickly than a traditional keyboard and mouse. As our population ages, the levels of disability increase and this means more and more people will have difficulty accessing services in the traditional way. Providing an alternative accessible digital solution, will ensure disabled people can continue to be independent. You may also discover a significant new market when you develop apps that suit these users.

Accessibility works in tandem with usability and ensures that there is an excellent user experience for all users of your apps regardless of whether they have a disability or

¹ www.who.int/mediacentre/factsheets/fs352/en

² webaim.org/projects/screenreadersurvey5/

not. Everyone wants to be able to complete the task they are undertaking on an app as quickly and easily as possible and if they have difficulties doing this, they are likely to go elsewhere. Embedding accessibility into your apps ensures an excellent experience for all.

Why Make your Apps Accessible?

Sometimes having access to services digitally enables a disabled person to use a service they may otherwise be unable to access. For example, if they are unable to get out of the house to do their shopping or online banking, then providing accessible online services means they can access these services independently. It is important to recognize how important independent access to services is for people with disabilities.

There are lots of other reasons to make your apps accessible:

- Implementing accessibility can often improve overall usability: For example, if you ensure that every button and form element has appropriate label, that is helpful to everyone, not just those with disabilities as the user will know how to interact with it.
- It just makes good business sense: For example, people with disabilities have spending power and if they find an accessible app that works for them they will not only use it, they will also tell others.
- Where accessible solutions are mandated by legislation, your app may be the only option for that business to realistically use: For example, your app may be able to tap into government funded market sectors such as education where legislation, such as Section 508 of the Rehabilitation Act in the US, may mandate an accessible solution.

- Access to goods and services for all is the law in many countries: For example in the UK the Equality Act 2010 requires there to be access to goods and services for everyone and this does include services which are provided via an electronic means such as websites and apps. Public bodies also have an anticipatory duty to ensure their services are accessible, so they cannot consider accessibility as an afterthought.
- The organization that the app is being developed for may have a corporate social responsibility (CSR) statement or program: For example, web and app accessibility provides social inclusion for the people with disabilities which is a primary aspect of corporate social responsibility.
- Mobile platforms from Apple, Google and Microsoft leverage their accessibility APIs for UI automation testing: Making your app accessible can make automated testing easier.

What Accessibility Features?

As many of your potential users may have a disability this can make it more difficult for them to use a mobile phone and related apps. Disabilities could include various levels of sight or hearing impairment, cognitive disabilities or learning difficulties, physical disabilities, dexterity issues, and so on.

Many of these users rely on third-party software to assist them in using their device. This software is sometimes called Assistive Technology, and includes different utilities depending on the type of disability. Traditionally these types of software or utilities have had to be 'added on' to a mainstream device, often at high cost, in order to make them accessible or easier to use for someone with a disability. Some smartphones and tablets now provide enough capabilities that some users with

disabilities can use the devices without needing to pay for extra Assistive Technology. What is offered depends on the platform and the version of the OS. However - to work - these features may need the app to be designed and implemented to support them.

- **Partially sighted users** - Someone who is partially sighted benefits from being able to change the font size, font style and use of bold and color contrast too. iOS, Android, Blackberry and Windows Phone offer various options to change these in the Settings. As well as the universal 'pinch to zoom' feature, iOS, Android, Blackberry and Windows Phone offer a magnification or zoom feature, which enlarges a section of the screen and keeps this magnification level when moving throughout the phone. This has unique gestures associated with it.
- **Blind users** - Someone who is blind has to have information on the screen and navigation around the screen announced to them in synthetic speech. This is often called a 'screen reader'. iOS was the first OS to offer a screen reader built-in and it is called 'VoiceOver'. Android offers 'Talkback' (fully featured since Android 4.1 Jelly Bean) which is fast catching up in popularity with the blind community as it is constantly improving. Windows Phone has just delivered Narrator in Windows Phone 8.1, but it is currently not at the point where it can be used to fully access the phone. Blackberry offers a screen reader with limited functionality in only few devices.
- **Users with hearing loss** - Someone with a hearing impairment will often make use of a smartphone that is hearing aid compatible and offers features as iOS does such as 'LED Flash for Alerts' or 'Phone Noise Cancellation'. There are also options in settings for iOS and Android to switch on

subtitles and captioning. Making use of vibrate for alerts is also helpful. A number of phones also provide support for hearing aids and TTY devices. A TTY device allows people who have hearing loss or who are speech impaired to type messages to anyone else who has a TTY, using a telephone line.

- **Users with physical disabilities** - If a user has a motor impairment, they may well be using a third party hardware product to access the phone, such as a switch as some devices do support this. Alternatively they could be making use of voice recognition to access the device.
- **Users with a learning disability** - If a user has a cognitive impairment or learning difficulty, then depending on what the disability is, they may make use of the features in the settings that a partially sighted user does. Especially something like color options. Other users may make more use of voice recognition.

For people with disabilities, their overall experience is affected by how well an app works with the assistive technology. As these features are built into the OS and can be switched on in the settings, it is important that as a developer you consider that they may be used with your app and ensure you test for this.

As screen readers and screen magnification in the OS makes use of their own gestures, gestures in the app may be affected when screen readers or magnification are enabled. For example a screen reader user can navigate a screen using left and right swipes or by exploring the screen by moving their finger across the screen of the device in a consistent movement. As they undertake a swipe, or encounter something underneath their finger, the item is announced. So an item is selected by tapping once and opened by tapping twice. When using screen

magnification, depending on the OS, they may need to use a three finger gesture. Including testing early on with accessibility features ensures that these gestures are supported by the app and that any redesign can happen before it impacts on users.

One of the best ways to learn more about these features is to switch them on and try them for yourself in different apps.

App Design Guidelines

The accessibility APIs look for text in specific attributes of standard UI elements. Screen readers used by blind people, such as VoiceOver and TalkBack, transform the text into synthetic speech which the user listens to. The screen reader software may also determine the type of control and related attributes to help provide the user with more contextual information, particularly if no text is available. It is important that the user understands what the label of the control is, what the control is and how to interact with it. In some instances there may also be a tooltip to give extra information.

Just as web developers make use of standards and guidelines such as WCAG 2.0 to make accessible websites, it is important that as app developers, you do the same. At present there is no de facto industry standard for app accessibility, although there are standards out there that can help.

The international standard, ISO 9241-171 ('The Ergonomics of Human-system Interaction: Guidance on Software Accessibility')³ is a helpful standard as it is platform agnostic. This covers elements of accessibility and usability for a wide range of software.

³ www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39080

The Royal National Institute of Blind People (RNIB)⁴ have created a pan-disability app standard and testing process based on their experience in this area of accessibility. Their standard for native apps also reflects on principles from ISO 9241-171. They provide consultancy and training for organizations and agencies in this area and have an accreditation badge that can be awarded to apps that, following an audit process are accessible.

The BBC have developed a set of BBC Mobile Accessibility Guidelines⁵ that they use internally for their mobile content. Their guidance covers mobile websites, hybrid and native apps. They state that "they are intended as a standard for BBC employees and suppliers to follow however they can also be referenced by anyone involved in mobile development".

Here are some of the principles that are helpful to be aware of when developing an app. If you stick to them, you will also give your app the best chance of interoperating with assistive technology that the user may be running in conjunction with your software:

APIs and UI Guidance

- Find out what accessibility features and APIs your platform has and follow best practice in leveraging those APIs if they exist.
- Use standard rather than custom UI elements where possible. This will ensure that if your platform has an accessibility infrastructure or acquires one in the future, your app is likely to be rendered accessibly to your users

⁴ www.rnib-business.org.uk

⁵ www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile_access.shtml

- Use the Accessibility API for your platform, if there is one. This will enable you to make custom UI elements more accessible and will mean less work on your part across your whole app.
- Follow the standard UI guidelines on your platform. This enhances consistency and may mean a more accessible design by default
- The user should be able to apply their preference settings that the OS provides, such as accessibility settings.

Navigation

- Navigation should be logical and consistent. For example if a back button is provided on each screen it should be located in the same place on every screen and consistently labeled.
- Support programmatic navigation of your UI. This will not only enable your apps to be used with an external keyboard but will enhance the accessibility of your app on platforms such as Android where navigation may be performed by a trackball or virtual d-pad.

User elements

- All user elements should be discoverable and operable via assistive technology, unless it is clear they are not required
- Where a user element has a status associated with it, that status should also be available to be read by assistive technology. For example if a toggle button is 'on' this should be announced by the screen reader. If the status changes, that should also be announced.
- Ensure touch screen targets are a reasonable size to ensure everyone can easily select them

Labeling

- All elements, including form elements, buttons, icons and so on, should be labeled visually and programmatically with a short and descriptive name. The label should also be adjacent to the element it relates to.
- Each screen should have a unique descriptive name that relates to its content and aids navigation

Colors and Fonts

- Ensure there is a good contrast between background and foreground colors
- Avoid using color as the only means of differentiating an action. A color-blind user will not be able to identify errors if they are asked to correct the fields which are highlighted in red for example.
- Consider the size of your smallest font. Is it reasonable that most people could read it without difficulty?

Notifications

- Error messages, notifications and alerts should be identifiable and clear.
- Ensure that error messages, notifications and alerts are not provided by colour/haptic/audible output alone. For example, someone with hearing loss will not recognize audible notifications.

Testing

- Don't forget to test your app on the target device with the assistive technology built-in to the OS
- Ensure your user testing includes people with disabilities too!

Apple, Google and Microsoft, have increased the importance of their respective Accessibility support by using the Accessibility interface to underpin their GUI test automation frameworks. This provides another incentive for developers to consider designing their apps to be more accessible.

Looking at the different mobile platforms more closely, it becomes obvious that they differ largely regarding their APIs, but they are starting to implement a lot of the same accessibility features.

Custom Controls and Elements

If you are using custom UI elements in your app, then, those platforms that have an Accessibility API enable you to make your custom controls accessible. You do this by exposing the control to assistive technology running on the device so that it can interrogate the properties of the control and render it accessibly.

You can get more information about this process on Android from the Google IO 2012 presentations⁶

If you are a member of the Apple developer program, then take a look at their accessibility video presentations from WWDC 2014 available in the iOS Developer Center⁷.

⁶ youtube.com/watch?v=q3HliaMjL38 and youtube.com/watch?v=ld7kZRpMGb8

⁷ developer.apple.com/wwdc/videos

Android App Accessibility

The latest major version of Android, Version 5 (Lollipop), has continued to improve the accessibility support with new accessibility APIs. Accessibility was really first a realistic proposition with Android 4.1 (Jellybean) and it is much improved since then.

Moving on from accessible live regions and closed captioning support in version 4.0, in 5.0 there are also experimental features in the settings which include color inversion and high contrast text. Android Services⁸

Accessibility features in Android 5.0 include (but are not limited to) things such as:

- **TalkBack** - speech output for blind users
- **Switch Access** - for those with physical disabilities who prefer to access apps using a hardware device
- **Captions support** - providing captions or subtitles for those with hearing loss
- **Magnification gestures** - zoom style magnification for partially sighted users
- **Large text and High Contrast Text** (Experimental) - for partially sighted users and some users with learning difficulties
- **Touch and hold delay** - for users with motor control issues
- **Color inversion** (Experimental) - for partially sighted users and some users with learning disabilities who prefer an inverted color palette
- **Voice input** - for users with learning disabilities and physical disabilities who wish to control the phone using their voice

⁸ developer.android.com/about/versions/lollipop.html

There are some helpful resources in the Support Library⁹ which also includes ways to improve the accessibility of custom views.

For specifics on how to use the Android accessibility API along with details of best practice in Android accessibility, please see Google's document entitled Making Applications Accessible¹⁰.

You will also find more examples in the training area of the developer documentation in a section entitled Implementing Accessibility¹¹. Testing the Accessibility is also covered online¹².

For more information about Android accessibility including how to use the text to speech API, see the Eyes-Free project¹³.

BlackBerry App Accessibility

Currently the Blackberry OS has some features for accessibility which are helpful for people with various disabilities. The features that they offer are more limited, however they do offer support for TTY for people with hearing loss. As there is only a screen reader available for limited Blackberry devices and it is not as developed as VoiceOver for iOS and TalkBack for Android, Blackberry devices are being considered by very few blind people at present.

For Blackberry 7 and Blackberry 10 devices, the Blackberry Screen Reader¹⁴ is only available for a very limited number of

⁹ developer.android.com/tools/support-library/index.html

¹⁰ developer.android.com/guide/topics/ui/accessibility/apps.html

¹¹ developer.android.com/training/accessibility/index.html

¹² developer.android.com/tools/testing/testing_accessibility.html

¹³ code.google.com/p/eyes-free

¹⁴ mobileapps.blackberry.com/devicesoftware/entry.do?code=bsr

devices. It comes pre-installed on the Blackberry 10 devices and can be downloaded for other supported devices. You can find out which are currently supported on the Blackberry Accessibility Website¹⁵

Blackberry 10 provides various accessibility settings to enable users to tailor their device. These include but are not limited to:

- **Screen Reader** - This turns text into synthetic speech for users with little or no useful vision. However it is only available on the BlackBerry Z30
- **Magnify Mode** - this enables the user to increase and decrease the magnification on the screen of text and elements.
- **Closed Captions** - Closed captions, or subtitles are helpful in video content for those with hearing loss
- **Display Settings** - these options give the user the chance to change the text and colors on the screen. This is helpful for people with learning disabilities and a partially sighted users
- **TTY Settings** - This is for users with hearing loss who want to use a teletypewriter with their device
- **Hearing aid support** - this is available on some phones

Documentation on creating accessible Blackberry 10 apps can be found at

Accessibility features and best practices - BlackBerry Native¹⁶

If you are designing for Blackberry 10 there are also devel-

¹⁵ us.blackberry.com/legal/accessibility.html

¹⁶ developer.blackberry.com/native/documentation/cascades/best_practices/accessibility/accessibility_features_best_practices.html

oper resources that include some design guidelines. Blackberry 10 Design Guidelines¹⁷

If you are developing for BB OS 7.1 you will find extensive information about the use of their accessibility API and many hints on accessible UI design on their website for developers¹⁸.

iOS App Accessibility

Apple were the first company to embed accessibility features directly into the OS. Because of this the support for accessibility in iOS is a little better than in Android, although Android is fast catching up.

Some of the accessibility features in iOS 8 include, but are not limited to:

- **VoiceOver** - a screen reader. It speaks the objects and text on screen, enabling your app to be used by people who are blind
- **Zoom** - This magnifies the entire contents of the screen
- **Invert Colours** - This inverts the colors on the display, which helps many people who need the contrast of black and white but find a white background emits too much light
- **Larger Text and Bold text** - this can help a broad range of people from those who use glasses, through to partially sighted people and those with learning difficulties
- **Increase Contrast** - this improves the contrast between the background and the foreground
- **Captioning and subtitles** - for people with hearing loss

¹⁷ developer.blackberry.com/devzone/design/bb10/accessibility.html

¹⁸ developer.blackberry.com/java/documentation/intro_accessibility_1984611_11.html

- **Audible, visible and vibrating alerts** - to enable people to choose what works best for them for notifications
- **Voice Control and Siri** - This enables users to make phone calls and operate various other features of their phone by using voice commands. This can be helpful for a broad range of people including those with motor control issues, learning difficulties and vision loss
- **Hearing aid compatibility** - for people with hearing loss
- **Switch control** - for those with physical disabilities who wish to access the app using a third party hardware device
- **Guided Access** - This is helpful in education, or just where someone wants to limit what is accessible on the screen to a user

If you are working on iOS, make sure to follow Apple's accessibility guidelines¹⁹. These guidelines detail the API and provide an excellent source of hints and tips for maximising the user experience with your apps.

Apple also provide some helpful guidance on testing the accessibility on your app with Voiceover²⁰

Windows Phone 8 App Accessibility

It is fair to say that Microsoft have been playing catch up with iOS and Android as far as accessibility features go. They rolled out some good support for magnification, text enlargement and changing of colours and in Windows Phone 8.1 things have

¹⁹ developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility

²⁰ developer.apple.com/library/ios/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityonYourDevicewithVoiceOver/TestAccessibilityonYourDevicewithVoiceOver.html

moved on again. There is now a screen reader called Narrator in Windows Phone 8.1, albeit still in beta. It reads out text in synthetic speech and like other phone screen readers, it makes use of its own specific gestures. It is designed for users with little or no vision but it still needs enhancing to be as comprehensive as the iOS and Android offerings. It can only be used with some core functionality and navigation functionality and is not as fully featured as other phone screen readers. At the time of writing it is only available with the US English display, keyboard and speech language settings. However, this is certainly one to watch for the future!

Some of the accessibility features on Windows Phone 8/8.1 include but are not limited to:

- **Narrator** - this screen reader is in beta, has limited functionality and is only available on 8.1
- **Text Size** - the size of text can be enlarged to aid those with learning difficulties or users who are partially sighted
- **High Contrast Theme** - this theme changes text to black and white and provides a solid background behind words that would otherwise be on top of pictures. This is helpful for partially sighted users and some users with learning disabilities
- **Screen Magnifier** - this feature is for partially sighted people who wish to magnify the text on the screen and change the zoom level. It has its own gestures.
- **Speech for phone accessibility** - the user can make calls, search the web, open apps or listen to text messages with Speech. This is helpful for a broad range of people in different situations, including those with motor impairments, learning disabilities and those with a visual impairment.
- **Customize browser captions** - It is possible to change the font size, color and background transparency of captions

in Internet Explorer and also apps that make use of the browser. This is helpful for people with hearing loss that may also have some vision loss

- **TTY support** - this device allows people who have hearing loss or who are speech impaired to type messages to anyone else who has a TTY, using a telephone line.
- **Cortana** - is the 'personal assistant' that is only available on Windows Phone 8.1. This is a main feature for all users, but will be helpful for those with disabilities too as it is speech activated.

The Accessibility for Windows Runtime Apps documentation²¹ provides support whether you are developing in C#/VB/C++ and XAML or JavaScript and HTML.

Microsoft has Guidelines for Designing Accessible Apps²² which is a really useful document. It relates to the relevant API information and if you really have to use custom controls in XAML or HTML, it gives help on how to do this in an accessible way. It also picks up some of the other areas of external guidance that are useful. For example, if you are developing in HTML then it will be important to think about using Accessible Rich Internet Applications 1.0 (WAI-ARIA)²³ which is helpful for making more dynamic content accessible to screen readers.

Once you have tested the accessibility of your app²⁴, Microsoft uniquely allows you to declare your app as accessible²⁵ in the Windows store, allowing it to be discovered by those who are filtering for accessibility in their searches.

²¹ msdn.microsoft.com/en-us/library/windows/apps/xaml/dn263101.aspx

²² msdn.microsoft.com/en-us/library/windows/apps/hh700407.aspx

²³ www.w3.org/TR/wai-aria/markup

²⁴ msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994937.aspx

²⁵ msdn.microsoft.com/en-us/library/windows/apps/xaml/jj161016.aspx

Mobile Web App Accessibility

As mentioned earlier in the chapter, much has been written about web accessibility, but less has been written on accessibility relating to apps. This is also true of mobile website accessibility or web app accessibility. It is an area which has growing interest and the World Wide Web Consortium (W3C) have created a 'Web and Mobile Interest Group' to discuss the area and to identify what work needs to take place. There is also currently a 'Web Applications Working Group'. So the number of groups that relate to this area of work in the W3C are growing and they can provide helpful documentation and support.

W3C have published a state of play and roadmap document which lists Standards for Web Applications on Mobile²⁶

It is suggested by the W3C that anything that uses HTML and is web based should still follow the Web Content Accessibility Guidelines (WCAG) 2.0 while also referring to Mobile Web Best Practices (MWBP). So if you are a web content developer then these guidelines are a good place to start. You will also find Relationship between Mobile Web Best Practices (MWBP) and Web Content Accessibility Guidelines (WCAG)²⁷ a helpful resource.

If your app is intended to mimic a native app look and feel, then you should follow the guidelines mentioned above in this chapter.

As support of HTML 5 is increasingly adopted on the various mobile platforms, consider reading Mobile Web Application Best Practices²⁸ as this is likely to form the foundation of any

²⁶ www.w3.org/Mobile/mobile-web-app-state

²⁷ w3.org/TR/mwbp-wcag/

²⁸ w3.org/TR/mwabp

mobile web application accessibility standard that emerges in the future. One of the other key areas of guidance is Accessible Rich Internet Applications 1.0 (WAI-ARIA)²⁹, as it has been designed to ensure that more dynamic HTML functionality is accessible to screen readers.

An interesting area of work happening at the W3C is in the Independent User Interface (IndieUI) Working Group³⁰. The group states "Independent User Interface (IndieUI) is a way for user actions to be communicated to web applications and will make it easier for web applications to work in a wide range of contexts — different devices, different assistive technologies (AT), different user needs". This work is going to be very important for accessibility and device independence. It is worth looking at the documentation that they currently have out for comment.

²⁹ www.w3.org/TR/wai-aria

³⁰ www.w3.org/TR/indie-ui-context



Testing

There are many parallel worlds for mobile apps on mobile devices. Meanwhile mobile devices evolve incredibly rapidly compared to most worldly goods. Testing mobile apps needs to keep pace even as the pace of change continues to accelerate.

The fate of mobile apps hangs in the balance where users, like crowds in Amphitheaters back in Roman times, often make the ultimate decision of whether an app lives to fight another day, or dies. And similarly, unremarkable apps are likely to languish as a statistic in the App Store, negating much of the hard work involved in conceiving, nurturing and launching it. Furthermore, the stigma of a poor rating has a long half-life which is hard to recover from.

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

Beware of specifics

Platforms, networks, devices, and even firmware, are all specific. Any could cause problems for your applications. There are several ways to identify the effects of specifics, for instance a tester may notice differences in the performance of the app and the behaviour of the UI during interactive testing with different devices. QuizUp used automated tests which helped them find five significant issues in their Android app triggered by differences in the devices, and one bug specific to Android

4.0.4. The automated tests ran across 30 devices in 30 minutes which made multi-device testing practical and useful, rather than spending 60 hours trying to do manual testing of the app on 30 devices¹. You need to know about these specifics to be able to decide whether to address undesirable differences by modifying the app before it is launched.

Conversely, Mobile Analytics can help identify differences in various aspects including performance and power consumption when the app is being used by many users on the vast variety of their devices. Some compelling examples of differences in behaviour and on ways issues were addressed in a paper published by computer scientists from the University of Wisconsin².

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

Testing Needs Time - Plan Accordingly

Calculation Example

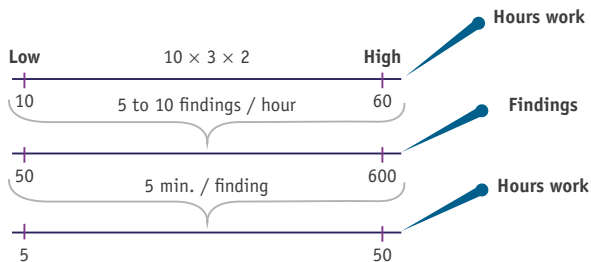
Let's assume your app has ten epics³) and every epic needs to be tested using between one and three perspectives (functionality, usability, user scenarios). If you test every epic with one perspective with one test case in one hour it takes ten hours to test the app. If you test every epic with one perspective with one test case per hour it will take ten hours to test the app. On the other hand if you test every epic using all three perspectives with two test cases, testing will take 60

1 blog.testmunk.com/quizup-mastering-android-device-fragmentation-automated-testing/

2 "Tale of Two Apps, available as a download at static.googleusercontent.com/media/research.google.com/en//pubs/archive/41590.pdf

3 also called user stories in this context (en.wikipedia.org/wiki/User_story)

hours. Typically, testing finds defects and other work worth reporting, which takes more time, where you also decide to spend time addressing some of the findings, for instance to fix a defect or otherwise improve the app. Let's assume that every executed test case results in 5 to 10 findings. The time needed for processing the defects lies between 5 to 50 hours work. So in the most positive scenario it takes 15 hours to test the app (only functionally) and in the more complex scenario it can take up-to 110 hours to test the app using three perspectives.



You may also need to factor in much more time to test the app on a variety of compatible devices, particularly for web apps and for Android native apps.

Continuous Testing

Continuous delivery needs continuous testing. Viable apps need to be updated on an ongoing basis in production. Updates may include fixes for new platform versions or device models, new functionality and other improvements. Therefore testing is not a one-off task; high quality apps benefit ongoing, optimized testing, including testing in production. Production testing includes testing engagement and validation as well as early detection of potential problems before they mushroom.

Manage your Testing Time

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

- **Reduce setup time:** Find ways to deploy apps quickly and efficiently. Implement mechanisms to provide the appropriate test data and configuration on both the mobile device and the relevant servers. Aim to have devices and systems 'ready to test'.
- **Reduce time needed for reporting & bug analysis:** Data, screenshots, and even video, can help make bugs easier and faster to investigate. Data can include logs, system configurations, network traffic, and runtime information.
- **Risk Analysis:** You can use the risk analysis to decide how and when to allocate testing effort. Risks are hard to determine accurately by the tester or developer alone; a joint effort from all the stakeholders of the mobile app can help to improve the risk analysis. Sometimes, the mobile app tester is the facilitator in getting the product risk analysis in place.

Involve others in your testing

People outside your immediate project team can help with the testing. For instance, other employees can participate in bug-bashes⁴, Crowdsourcing⁵ involves outsiders paid to help

⁴ en.wikipedia.org/wiki/Bug_bash

⁵ service providers include Applause.com, PassBrains.com, Mob4Hire.com and TestBirds.de

test your app, and Alpha & Beta testing provides early access to new releases to subsets of users.

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

These approaches can augment your other testing, we do not recommend using them as your only formal testing. To get good results you will need to devote some of your time and effort to defining the tests you want them to run, and to working with the company to review the results, et cetera.

Effective Testing Practices

Testing, like other competencies, can be improved by applying various techniques and practices. Some of these need to be applied when developing your mobile app, such as testability, others apply when creating your tests, and others still when you perform your testing. Testdroid offers a good checklist⁶ on getting the right testing expertise into your team.

Mnemonics summarizing Testing Heuristics

Heuristics are fallible guidelines, or rules-of-thumb, that tend to be useful. Several have been created specifically to help test mobile apps and some use mnemonics to help you consider particular aspects of software. Each letter is the initial letter of a word representing a key word.

⁶ testdroid.com/testdroid/6336/get-the-superb-expertise-in-your-testingqa-team

- **I SLICED UP FUN⁷**: Input (Test the application changing its orientation (horizontal/vertical) and trying out all the inputs including keyboard, gestures etc.), Store (Use appstore guidelines as a source for testing ideas), Location (Test on the move and check for localisation issues), Interaction/Interruption (See how your app interacts with other programs, particularly built-in, native apps), Communication (Observe your app's behaviour when receiving calls, e-mails, etc.), Ergonomics (Search for problem areas in interaction, e.g. small fonts), Data (Test handling of special characters, different languages, external media feeds, large files of different formats, notifications), Usability (Look for any user actions that are awkward, confusing, or slow), Platform (Test on different OS versions), Function (Verify that all features are implemented and that they work the way they are supposed to), User Scenarios (Create testing scenarios for concrete types of users), Network (Test under different and changing network conditions)
- **COP FLUNG GUN⁸** summarizes similar aspects under Communication, Orientation, Platform, Function, Location, User Scenarios, Network, Gestures, Guidelines, Updates, Notifications

Implementing Testability

Start designing and implementing ways to test your app during its development already; this applies especially for automated testing. For example, using techniques such as Dependency Injection in your code enables you to replace real servers (slow and flaky) with mock servers (controllable and fast).

⁷ kohl.ca/articles/ISLICEDUPFUN.pdf

⁸ moolya.com/blogs/2014/05/34/COP-FLUNG-GUN-MODEL

Use unique, clear identifiers for key UI elements. If you keep identifiers unchanged your tests require less maintenance.

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

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

- **Combination testing:** is a established technique to find bugs caused by combinations of parameters efficiently, in far fewer tests than trying all permutations. A popular technique is called All Pairs Testing
en.wikipedia.org/wiki/All-pairs_testing

Tours for Exploratory Testing

A tour is a type of exploratory testing, a way to more structure the exploratory test sessions. Tours help you focus your testing, Cem Kaner describes a tour as "a directed search through the program. Find all the capabilities. Find all the claims about the product. Find all the variables. Find all the intended benefits. Find all the ways to get from A to B. Find all the X. Or maybe not ALL, but find a bunch"⁹. With the combination of different tours in different perspectives (see the I SLICED UP FUN heuristics) coverage and test depth can be chosen.

⁹ kaner.com/?p=96, also see developsense.com/blog/2009/04/of-testing-tours-and-dashboards/

Examples of Tours¹⁰ include:

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

Learn more about this and other methods of exploratory testing in James Whittaker's book "Exploratory Software Testing"¹¹.

Testing on Various Devices

Physical and Virtual Devices

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

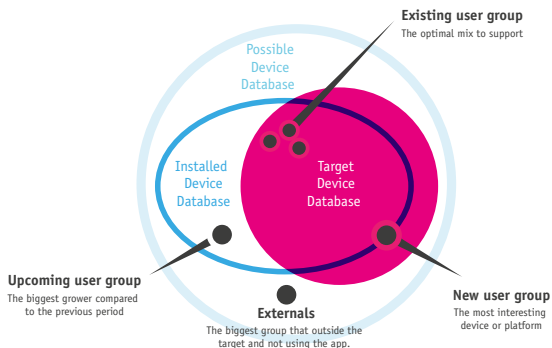
Virtual devices are generally free and immediately available to install and use. Some platforms, including Android, allows you to create custom devices, for instance with a new screen resolution, which you can use for testing your apps even before suitable hardware is available. They can provide rough-and-ready testing of your applications. Key differences

¹⁰ from michaeldkelly.com/blog/2005/9/20/touring-heuristic.html

¹¹ www.amazon.com/Exploratory-Software-Testing-Tricks-Techniques/dp/0321636414

include: performance, security, and how we interact with them compared to physical devices. These differences may affect the validity of some test results.

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



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

Here are some examples of areas to test on physical devices:

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

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

Remote Devices

If you do not have physical devices at hand or if you need to test your application on other networks, especially abroad and for other locales, then one of the 'remote device services' might help you. They can help extend the breadth and depth of your testing at little or no cost.

Several manufacturers provide this service free-of-charge for

a subset of their phone models to registered software developers. Both Nokia¹² for their platforms; and Samsung¹³ (for Android and Tizen) provide restricted but free daily access.

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

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

Beware of privacy and confidentiality when using shared devices.

Test Automation

Automated tests can help you maintain and improve your velocity, your speed of delivering features, by providing early feedback of problems. To do so, they need to be well-designed and implemented. Otherwise you risk doubling your workload to maintain a mess of broken and unreliable automated tests as well as a broken and an unreliable app. Good automated tests mimic good software development practices, for instance using Design Patterns¹⁴, modularity, performing code reviews, et cetera.

Testing mobile applications effectively can be complex and challenging where you need to combine automated and interactive testing across a range of devices. Thankfully,

¹² developer.nokia.com/Devices/Remote_device_access/

¹³ developer.samsung.com/remotetestlab/rtlDeviceList.action

¹⁴ en.wikipedia.org/wiki/Design_Patterns

several of the major mobile development platforms include test automation in the core tools, including Android and iOS. And cross-platform test automation tools are available for popular platforms; some are free-of-charge and open-source, others are commercial.

It is important to assess the longevity and vitality of the test automation tools you plan to use, otherwise you may be saddled with unsupported test automation code. Test automation tools provided as part of the development SDK are worth considering. They are generally free, inherently available for the particular platform, and are supported by massive companies.

BDD Test Automation

BDD stands for Behavior-Driven Development¹⁵ where the behavior is described in formatted text files that can be run as automated tests. The format of the tests are intended to be readable and understandable by anyone involved with the software project. They can be written in virtually any human language, for instance Japanese¹⁶, and they use a consistent, simple structure with statements such as **Given, When, Then** to structure the test scripts.

There are various BDD frameworks available to test mobile apps. These include:

- **Calabash** for Android and iOS: github.com/calabash
- **Frank** for iOS: www.testingwithfrank.com
- **RoboGerk** for Android: github.com/leandog/RoboGherk
- **Zucchini** for iOS: www.zucchiniframework.org

¹⁵ en.wikipedia.org/wiki/Behavior-driven_development

¹⁶ github.com/cucumber/cucumber/tree/master/examples/i18n/ja

and various implementations that integrate with Selenium-WebDriver for testing web apps, including web apps on iOS and Android.

Often, custom 'step-definitions' (small scripts that interact with the app being tested) need to be written by someone with coding skills.

GUI Test Automation

GUI test automation is where automated tests interact with the app via the Graphical User Interface (GUI). It is one of the elixirs of the testing industry, many have tried but few have succeeded in creating useful and viable GUI test automation for mobile applications. One of the main reasons why GUI test automation is so challenging is that the User Interface is subject to significant changes which may break the way automated tests interact with the app.

For the tests to be effective in the longer term, and as the app changes, developers need to design, implement and support the labels and other hooks used by the automated GUI tests. Both Apple, with UI Automation¹⁷, and more recently Android¹⁸ use the Accessibility label assigned to UI elements as the de-facto interface for UI automation.

Some commercial companies have opensourced their tools, e.g. GorillaLogic's MonkeyTalk¹⁹ and Xamarin's Calabash²⁰. These tools aim to provide cross-platform support particularly

¹⁷ developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/UsingtheAutomationInstrument/UsingtheAutomationInstrument.html

¹⁸ developer.android.com/tools/testing/testing_ui.html

¹⁹ gorillalogic.com/testing-tools/monkeytalk

²⁰ github.com/calabash

for Android and iOS. Other successful opensource frameworks include Robotium²¹ and Frank²².

Unit Testing

Unit testing involves writing automated tests that test small chunks of code, typically only a few lines of source code. Generally they should be written by the same developer who writes the source code for the app as they reflect how those individual chunks are expected to behave.

Unit tests have a long pedigree in software development, where JUnit²³ has spawned similar frameworks for virtually all of the programming languages used to develop mobile apps.

Unit tests are only one aspect of automated testing, they are not sufficient to prove the app works. They help developers to understand what individual pieces of the software is expected to do. Additional testing, including other forms of automated tests can help to increase your confidence in the app.

Testing Through The Five Phases of an App's Lifecycle

Software is developed in phases, which are called steps in the life cycle. A mobile app tester can be part of the development team, but can also be responsible to facilitate the user experience tests in production. Depending on which phase(s) you are involved in in the life cycle, there are different tasks to be performed. For example when joining a development team the task can be the analysis of the error in the log files on a de-

²¹ code.google.com/p/robotium

²² testingwithfrank.com

²³ en.wikipedia.org/wiki/JUnit

vice. When joining a beta test phase a task can be the analysis of usability tests results like recording movies. The complete lifecycle of a mobile app fits into 5 phases: implementation, verification, launch, engagement and validation. Testing applies to each phase. Some of the decisions made for earlier stages can affect your testing in later stages. For instance, if you decide you want automated system tests in the first phase they will be easier to implement in subsequent phases.

Phase 1: Implementation

This includes design, code, unit tests, and build tasks. Traditionally testers are not involved in these tasks; however good testing here can materially improve the quality and success of the app by helping us to make sure the implementation is done well.

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

- Do you use test-driven development (TDD)?
- Do you write unit tests even if we are not using TDD?
- Will you have automated system tests? If so, how will you facilitate these automated system tests? For instance by adding suitable labels to key objects in the UI.
- How will you validate your apps? For instance, through the use of Mobile Analytics? Crash reporting? Feedback from users?

Question the design. You want to make sure it fulfills the intended purposes; you also want to avoid making serious mistakes. Phillip Armour's paper on five orders of ignorance²⁴ is a great resource to help structure your approach.

²⁴ www-plan.cs.colorado.edu/diwan/3308-07/p17-armour.pdf

Also consider how to improve the testability of your app at this stage so you can make your app easier to test effectively and efficiently. Practices, including unit tests and TDD apply to the implementation phase. Remember to test your build process and build scripts to ensure they are effective, reliable and efficient, otherwise you are likely to suffer the effects of poor builds throughout the life of the app.

Phase 2: Verification

This includes reviewing unit tests, internal installation, and system tests.

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

For apps that need installing you need ways to deploy them to specific devices for pre-release testing. Some platforms (including Android, iOS and Windows Phone) need phones to be configured so development apps can be installed. You also need to decide which phones to test the app on. For instance, it is wise to test the app on each suitable version of the mobile platform. For iOS this may only include the latest releases. For Android you also need to consider low end devices that are still being sold with old Android versions and might never be updated.

You will also want to test different form-factors of devices; for instance where the ratio of the screen dimensions differ. The iPhone 5's and 6's new screen dimensions exposed lots of UI bugs. Android developers are well aware of the many issues different screen sizes can trigger.

System tests are often performed interactively, by testers. Consider evaluating test automation tools and frameworks for

some of your system tests. We will go into more detail later in this section.

You also want to consider how to make sure the app meets:

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

Phase 3: Launch

This includes pre-publication and publication.

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

Given these constraints it is worth extending your testing to include pre-publication checks of the app such as whether it is suitable for the set of targeted devices. The providers of the main platforms now publish guidelines to help you test your app will meet their submission criteria. These guidelines may help you even if you target other app stores.

Apple	developer.apple.com/appstore/resources/approval/guidelines.html
Android	developer.android.com/distribute/googleplay/publish/preparing.html#core-app-quality
Windows Phone	msdn.microsoft.com/en-us/library/windowsphone/develop/hh394032
BlackBerry	developer.blackberry.com/devzone/appworld/tips_for_app_approval.html

Phase 4: Engagement

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

With a continuous flow of new releases to production, a proactive approach is needed to monitor the engagement aspects.

Phase 5: Validation

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

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

Problem detection and reporting. These may include your own code, third-party utilities, and online services.

Mobile Analytics. Does the data being collected make sense? What anomalies are there in the reported data? What is the latency in getting the results, et cetera?

Testing Tools

- **Application Ressource Optimizer (ARO)** by AT&T: developer.att.com/application-resource-optimizer, open source project at github.com/attdevsupport/ARO
- **Fiddler** by Telerik: telerik.com/fiddler enables you to view and modify network traffic between your mobile device and the network.
- **Genymotion**: genymotion.com, a faster and more capable Android emulator, for instance to control sensor values.
- **iFunbox**: i-funbox.com/ helps to install apps and manage files on iOS devices.
- **iTools**: itools.en.uptodown.com/, an app configuration tool for iOS, including monitoring cache and realtime log files.

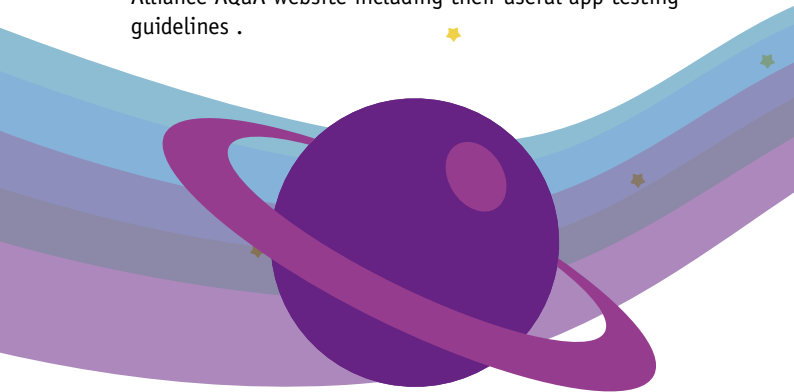


Learn More

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

- testdroid.com/blog, a fertile blog on various topics including testing mobile apps. They also have a series on testing mobile games²⁵.
- appqualitybook.com/, the website about Jason Arbon's interesting book based on the experiences of testing and analysing vast numbers of mobile apps.
- kohl.ca, Jonathan Kohl's website which includes links to his acclaimed book on testing mobile apps and other articles.
- slideshare.net/karennjohnson/kn-johnson-2012-heuristics-mnemonics, Karen Johnson's presentation on testing heuristics and applying SFD POT to mobile testing.
- appqualityalliance.org/resources, the official App Quality Alliance AQuA website including their useful app testing guidelines .

²⁵ testdroid.com/testdroid/7790/best-practices-in-mobile-game-testing







Monetization

Finally you have finished your app or mobile website and polished it as a result of beta testing feedback. Assuming you are not developing as a hobby, for branding exposure, et cetera, now it is time to make some money. But how do you do that, what are your options?

In general, you have the following monetization options:

- **Pay per download:** Sell your app per download
- **In-app payment:** Add payment options into your app
- **Mobile advertising:** Earn money from advertising
- **Sponsorships:** Receive money for each user signing up to your sponsor
- **Revenue sharing:** Earn revenue from operator services originating in your app
- **Indirect sales:** Affiliates, data reporting and physical goods among others
- **Component marketplace:** Sell components or a white-label version of your app to other developers

When you come to planning your own development, determining the monetization business model should be one of the key elements of your early design as it might affect the functional and technical behavior of the app. Five strategies to monetize your mobile app¹ is an excellent article on how to design the financial aspects so they don't annoy users or lose the revenues you hoped to receive.

¹ medium.com/@signored/dont-fall-below-the-app-poverty-line-9b800a214e4a

Pay Per Download

Using pay per download (PPD) your app is sold once to each user as they download and install it on their phone. Payment can be handled by an app store, mobile operator, or you can setup a mechanism yourself. Once the most popular and profitable monetization method, today it is only used by a minority of developers. Gartner predicts that by 2017 nearly 95% of downloads will be for free apps, up from approximately 90% today².

When your app is distributed in an app store, the store will handle the payment mechanism for you. In return the store takes a revenue share (typically 30%) on all sales. In most cases stores offer a matrix of fixed price points by country and currency (\$0.99, EUR 0.79, \$3 etc) to choose from when pricing your app.

Payment for downloaded apps is generally handled in one of two ways: operator billing or credit-card payments.

Operator billing enables your customers to pay for your app by just confirming that the sale will be charged to their mobile phone bill or by sending a Premium SMS. In some cases, operator billing is handled by an app store (such as Google Play, which supports operator billing for a number of carriers around the world). In other cases, it can be implemented directly by the developer.

Each operator will take a revenue share of the sale price (typically 30% to 65%, but some operators can take up to 95%), and, if you use one, an aggregator will take its share too. Security (how you prevent the copying of your app) and manageability are common issues with the PPD model, but

2 www.businessweek.com/articles/2013-09-19/the-profitable-future-of-free-mobile-apps

in some scenarios it might be the only monetization option. Operator billing can be quite difficult to handle on your own, particularly if you want to sell in several countries, as you need to sign contracts with each operator in each country. For unknown reasons some operators, like Vodafone, seem to remove operator billing as an option for Android Play in some key markets, like UK and Germany. Possibly because better alternatives, like local mobile bank payments become available.

It is worth noting that most of the vendor app stores are pursuing operator billing agreements, with Nokia Store having good coverage with operator billing available in 60 countries for both their legacy Nokia Store³ and Microsoft's Windows Phone Marketplace. Google and Blackberry have similar options. The principal reason they are doing this is that typically, when users have a choice of credit card and operator billing methods users show a significant preference for operator billing (Nokia says its research has shown up to a 10x increase in revenues over credit card payments). Nokia, at least, also insulates developers from the variation in operator share, offering developers a fixed 70% of billing revenue.

Credit-card billing is used by Apple, Google (in some cases), Amazon and other stores. Apple has required iPhone users to provide credit-card data at registration for many years, and Google now requires this as well for Android users. Having this information entered before purchase is, according to analysts, a key differentiator for higher monthly per app revenue.

The last payment option is to create your own website and

³ on current Nokia feature phones, 99% of app revenue comes through operator billing: developer.nokia.com/nokia-x/opportunity/monetization

implement a payment mechanism through that, such as PayPal mobile, dial-in to premium landlines⁴ or others.

Using PPD can typically be implemented with no special design or coding requirements for your app and for starters we would recommend using the app store billing options as it involves minimal setup costs and minor administrative overhead.

For each form of payment it is important to determine price elasticity of demand PED⁵. Increasing the price does not necessarily mean higher total revenue (and vice versa), your price needs to match expectations of your user base.

In-App Payment

In-app payment (IAP) is a way to charge for specific actions or assets within your application. A very basic use might be to enable the one-off purchase of your application after a trial period — which may garner more sales than PPD if you feel the features of your application justify a higher price point. Alternatively, you can offer the basic features of your application for free, but charge for premium content (videos, virtual credits, premium information, additional features, removing ads and alike). Most app stores offer an in-app purchase option or you can implement your own payment mechanism. If you want to look at anything more than a one-off “full license” payment you have to think carefully about how, when and what your users will be willing to pay for and design your app accordingly.

Recurring In-app payments, also known as subscriptions, are offered by most platforms as well. These type of payments

⁴ daopay.com

⁵ en.wikipedia.org/wiki/Price_elasticity_of_demand

fit well when your app offers content that is regularly updated, such as online newspapers or digital magazines.

In-app purchases have become the leading monetization model in many markets, particularly among "freemium" games that use free distribution to get users hooked before turning them into buyers.

IAP is particularly popular in games (for features such as buying extra power, extra levels, virtual credits and alike) and can help achieve a larger install base as you can offer the basic application for free.

Distimo reported in 2014 that in-app purchases accounted for 79% of iOS revenue⁶.

If you target specific countries, be aware of different behaviour, e.g. in China the initial purchase is 99% of all revenue generated, while IAP is very low, while in the US it is the other way around.

It should also be obvious that you will need to design and develop your application to incorporate the in-app payment method. If your application is implemented across various platforms, you may need to implement a different mechanism for each platform (in addition to each app store, potentially).

As with PPD we would recommend that you start with the in-app purchasing mechanism offered by an app store, particularly as some of these can leverage operator billing services (such as Google Play) or utilize pre-existing credit-card information (such as Apple or Amazon), or with in-app payment offered directly by operators. From a user's perspective, this is the easiest and most convenient way to pay (one or two clicks, no need to enter credit card numbers, user names or other credentials), so developers can expect the highest user acceptance and conversion rates.

6 "2014 How the Most Successful Apps Monetize Globally" available at www.distimo.com/publications

Mobile Advertising

As is common on websites, you could decide to earn money by displaying advertisements. There are a number of players who offer tools to display mobile ads and it is the easiest way to make money on mobile browser applications. Admob.com, Buzzcity.com and inmobi.com (targeting games) are a few of the parties that offer mobile advertising. However because of the wide range of devices, countries and capabilities there are currently over 70 large mobile ad networks. Each network offers slightly different approaches and finding the one that monetizes your app's audience best may not be straightforward. There is no golden rule; you may have to experiment with a few to find the one that works best. However, for a quick start you might consider using a mobile ad aggregator, such as Madgic⁷, smaato⁸ or inneractive⁹ as they tend to bring you better earnings by combining and optimizing ads from 50+ mobile ad networks. Most aggregators can also operate as an Ad Exchange, providing Real Time Bidding (RTB), like a live auction where the price of each ad is determined at run-time.

Most ad networks take a 30% to 50% share of advertising revenue and aggregators another 15% to 20% on top of that, but even with those numbers aggregators are still more profitable than trying to integrate all separate ad networks yourself.

If your app is doing really well and has a large volume in a specific country you might consider selling ads directly to advertising agencies or brands (Premium advertising) or hire a media agency to do that for you.

Again many of the device vendors offer mobile advertising services as part of their app store offering and these mecha-

⁷ madgic.com

⁸ smaato.net

⁹ inner-active.com

nisms are also worth exploring. In some cases you may have to use the vendor's offering to be able to include your application in their store.

In-application advertising will require you to design and code your application carefully. Not only the display location of ads within your app needs to be considered with care, also the variations and opt-out mechanism. If adverts become too intrusive, users may abandon your app, while making the advertising too subtle will mean you gain little or no revenue. Relatively new compared to traditional banner advertising is interstitial advertising: This term is generally used to describe an ad that takes up the entire screen and typically has a "skip screen" button at the bottom. Other new ad formats include playable ads or rewarded ads, especially for games. It may require some experimentation to find the right level and positions in which to place adverts.

Sponsorships

The German startup Apponsor¹⁰ offers a new way of earning money without the need to display advertising or charge a download fee: The user gets your app for free and is prompted to sign-up for a newsletter of your sponsor. The sponsor will in return pay the developer an amount for each newsletter registration. Not to be confused with App Sponsors, companies who pay the development costs of your App in return for a stake, see also Apps Funder¹¹.

¹⁰ apponsor.com

¹¹ appsfunder.com

Indirect Sales

Another option is to use your application to drive sales elsewhere.

Here you usually offer your app or website for free and then use mechanisms such as:

1. **Affiliate programs:** Promote third party or your own paid apps within a free app. See also MobPartner¹². This can be considered a variation on mobile advertising
2. **Data reporting:** Track behavior and sell data to interested parties. Note that for privacy reasons you should not reveal any personal information, ensure all data is provided in anonymous, consolidated reports
3. **Virtual vs. real world:** Use your app as a marketing tool to sell goods in the real world. Typical examples are car apps, magazine apps and large brands such as McDonald's and Starbucks. Also coupon applications as Groupon often use this business model

There is nothing to stop you from combining this option with any of the other revenue generation options if you wish, but take care that you do not give the impression of overly-intrusive promotions.

Component Marketplace

A Component Marketplace (CMP) provides another opportunity for developers to monetize their products to other developers and earn money by selling software components or white-labelled apps. A software component is a building block piece

¹² mobpartner.com

of software, which provides a defined functionality, that is to be used by higher level software.

The typical question that comes up at this point is on how CMPs contrast to open source. As a user, open source is often free-of-charge. Source code must be provided and users have the right to modify the source code and distribute the derived work.

Some component providers require a license fee. They may provide full source code which enables the developer to debug into lower level code. Some CMPs support all models: Paid components with or without source code as well as free components with or without source code.

If you are a developer searching for a component, CMPs offer two major advantages: First, you do not have to open source your code just because you use software components. All open source comes with a license. Some licenses like the Apache are commercially friendly; others, such as AGPL and OSL, require you to open source your code that integrates with theirs. You might not want this. Secondly, CMPs provide an easy way to find and download components. You can spend days browsing open source repositories to find the right thing to use.

Component marketplaces have existed for decades now. The most prominent marketplace is for components for Visual Basic and .NET in the Windows community. Marketplaces such as componentOne and suppliers like Infragistics are well known in their domain. The idea of component marketplaces within the mobile arena is quite new. Deutsche Telekom's Developer Garden¹³ and ChupaMobile¹⁴ are relevant players in this domain.

¹³ www.developergarden.com/component-marketplace/

¹⁴ www.chupamobile.com

Choosing your Monetization Model

So with all these options what should your strategy be? It depends on your goals, let us look at a few:

- Are you convinced users will be willing to buy your app immediately? Then sell it as PPD for \$0.99, but beware while you might cash several thousand dollars per day it could easily be no more than a few hundred dollars per week if your assessment of your app is misplaced or the competition fierce. The Application Developer's Alliance recommends the PPD monetization method mainly for high-production apps, apps with barriers to entry and high-volume apps¹⁵. This includes games and apps for entertainment, productivity, navigation and news.
- Do you target a large user base? Consider distributing your application for free with in-app purchases, or with mobile advertising (you could even offer a premium ad-free version).
- Are you offering premium features at a premium price? Consider a time or feature limited trial application then use in-app purchasing to enable the purchase of a full version either permanently or for a period of time.
- Are you developing a game? Consider offering the app for free with in-app advertising or a basic version then use in-app purchasing to allow user to unlock new features, more levels, different vehicles or any extendable game asset.
- Is your mobile app an extension to your existing PC web shop or physical store? Offer the app for free and earn revenue from your products and services in the real world.

¹⁵ www.appdevelopersalliance.org/app-monetization

- Does your app contain content that is updated frequently, like digital magazines? Offer recurring in-app payments and make sure that visitors return.
- Do you offer physical goods, like a webshop app? Offer your App for free and make money on the margins of your customer's purchases.

Appstore Strategies

The flip side of revenue generation is marketing and promotion. The need might be obvious if you sell your application through your own website, but it is equally important when using a vendor's app store. Appstores are the curse and the blessing of mobile developers. On the bright side they give developers extended reach and potential sales exposure that would otherwise be very difficult to achieve. On the dark side the more popular ones now contain hundreds of thousands of apps, decreasing the potential to stand out from the crowd and be successful, leading many to compare the chances of appstore success to the odds of winning the lottery.

So, here are a few tips and tricks to help you raise your odds.

Strategies To Get High Rankings

The most important thing to understand about appstores is that they are distribution channels and not marketing machines. This means that while appstores are a great way to get your app onto users' devices, they are not going to market your app for you (unless you purchase premium positioning either through banners or list placings). You cannot rely on the app stores to pump up your downloads, unless you happen to get into a top-ten list. But do not play the lottery with your apps, have a strategy and plan to market your app.

We have asked many developers about the tactics that brought them the most attention and higher rankings in appstores.

Many answers came back and one common theme emerged: there is no silver bullet – you have to fire on all fronts! However it will help if you try to keep the following in mind:

- You need a kick ass app: it should be entertaining, easy to use and not buggy. Make sure you put it in the hands of users before you put it in a store.
- Polish your icons and images in the appstore, work on your app description, and carefully choose your keywords and category. If unsure of or unsatisfied with the results, experiment.
- Getting reviewed by bloggers and magazines is one of the best ways to get attention. In return some will be asking for money, some for exclusivity, and some for early access.
- Get (positive) reviews as quickly as possible. Call your friends and ask your users regularly for a review.
- If you are going to do any advertising, use a burst of advertising over a couple of days. This is much more effective than spending the same amount of money over 2 weeks, as it will help create a big spike, rather than a slow, gradual push.
- Do not rely on the traffic generated by people browsing the appstore, make sure you drive traffic to your app through your website, SEO and social media.

Multi-Store vs Single Store

With 120+ appstores available to developers, there are clearly many application distribution options. But the 20 minutes needed on average to submit an app to an appstore means you

could be spending a lot of time posting apps in obscure stores that achieve few downloads. This is why a majority of developers stick to only 1 or 2 stores, missing out on a potentially huge opportunity but getting a lot more time for the important things, like coding! So should you go multi-store or not?

Multi-store	Single store
The main platform appstores can have serious limitations, such as payment mechanisms, penetration in certain countries, content guidelines.	90%+ of smartphone users only use a single appstore, which tends to be the platform appstore shipping with the phone
Smaller stores give you more visibility options (featured app)	Your own website can bring you more traffic than appstores (especially if you have a well-known brand)
Smaller stores are more social media friendly than large ones.	Many smaller appstores scrape data from large stores, so your app may already be there.
Operators' stores have notoriously strict content guidelines and can be difficult to get in, particularly for some types of apps.	For non-niche content, operator or platform stores may offer enough exposure to not justify the extra effort of a multi-store strategy.
Smaller stores may offer a wider range of payment or business model options, or be available in many countries.	Some operators' stores have easier billing processes – such as direct billing to a user's mobile account -- leading to higher conversion rates.
Some developers report that 50% of their Android revenues come from outside of Android Market	iOS developers only need 1 appstore

The platform app stores should give you general coverage for users, but over time, it is in your interest to adapt your appstore strategy to match your targeted user base, and utilize the appstores that best reach it. This could mean using particular operator stores, stores popular in a specific country, or simply sticking with the platform stores. There are some third-party appstores with large audiences, such as the Amazon appstore for Android, which offers developers a number of ways to monetize their apps, such as PPD and in-app payments in several countries. Additionally, in some countries, there are locally popular appstores, such as AndroidPit¹⁶ in Germany, or one of the many China-specific Android stores.

What Can You Earn?

One of the most common developer questions is about how much money they can make with a mobile app. It is clear that some apps have made their developer's millionaires, while others will not be giving up their day job anytime soon. According to a 2013 research by Forbes.com¹⁷, most app developers are not generating enough revenue to break even with development costs and single platform developers confirmed it was not enough to support a standalone business.

According to VisionMobile's Business and Productivity apps research among over 6,000 mobile developers 50% of them are below the "app poverty line" of \$500 per app per month (67% in 2013)¹⁸.

inMobi claims that in Q3 2014, mobile games were the most profitable kind of mobile software when it comes to ad

¹⁶ androidpit.de

¹⁷ forbes.com/sites/tristanlouis/2013/08/10/how-much-do-average-apps-make

¹⁸ www.visionmobile.com/product/business-productivity-apps

revenue. According to their whitepaper¹⁹, the best working ads are rich media apps.

Finally, Gartner does not paint a very bright picture for the future neither: They predict that through 2018, less than 0.01 percent of consumer mobile apps will be considered a financial success by their developers²⁰.

Ultimately, what you can earn is about fulfilling a need and effective marketing. Experience suggests that apps which save the user money or time are most attractive (hotel discounts, coupons, free music and alike) followed by games (just look at the success of Angry Birds) and business tools (office document viewers, sync tools, backup tools and alike) but often the (revenue) success of a single app cannot be predicted. Success usually comes with a degree of experimentation and a lot of perseverance.

19 www.inmobi.com/insights/download/whitepapers/the-state-of-mobile-app-monetization-q3-2014

20 www.gewsroom/id/2648515





Epilogue

Thanks for reading this 15th edition of our Mobile Developer's Guide. We hope you have enjoyed reading it and that we helped you to clarify your options. Perhaps you are now ready to get involved in developing a mobile app or have discovered new options in the app business. We hope so. Please also get involved in the community and share your experiences and ideas with us and with others.

If you like to contribute to this guide, sponsor upcoming editions or if you are interested in getting previous editions of the book, please write to mdgg@enough.de. If you are using Twitter, you are invited to follow us on twitter.com/MobileDevGuide and spread the word about the project using the Twitter hashtag #mdgg.

You can of course also get this guide as an ebook- just check amazon.com, kobobooks.com or apple.com/iBooks. Alternatively you can download the pdf version on our website: www.enough.de/mdgg. At the time of writing we are also working on making the content of this book available as a website at www.mobiledevelopersguide.com, where you will hopefully very soon find new ways to get involved and submit your feedback.

The 16th edition can be expected for early 2016!

About the Authors

Anna Alfut

Anna started her professional life as Creative Designer. After discovering her passion for interface design she co-authored an app for iOS and Android platforms and consulted on multiple projects both on the agency and client side. Currently she works as UX designer for mobile. Apart from thinking through and drawing UIs she also does illustration and enjoys living in London.

www.alfutka.net


Cornelius Kwietniak / Enough Software

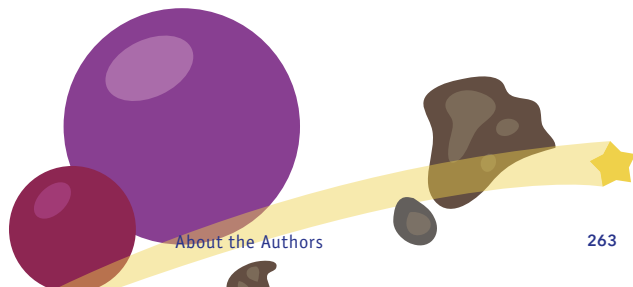
Cornelius specializes in graphic, UI, UX and visual design for mobile applications and other interactive technologies. He is also in charge of the layout and design of this guide. When not involved with something mobile, he loves to experiment with digital art and illustration.

www.enough.de

Davoc Bradley / MiraLife

Davoc has been working as a software engineer since 1999 specializing in architecture and design of high usage web and mobile systems. Currently he is CTO at MiraLife who specialize in providing web based and mobile software which aims to improve the lives of people suffering from dementia and other terminal illnesses. Davoc is also a keen musician, avid cricket fan and loves travelling.

 **Twitter:** @davocbradley



Marco Büttner / SciDev

Marco is 26 years old and has been a mobile developer since 2011. He is studying computer science at the Humboldt University of Berlin, works for idealo, and founded the mobile development project SciDev, which focuses on developing web apps for bada, Tizen and other emerging mobile platforms. He is well connected among the bada and Tizen community and always glad to share his know-how.

 **Twitter:** @scionbln www.scidevapps.de

Sally Cain / RNIB

Sally has worked at RNIB in the area of digital accessibility for more than 15 years. She believes passionately in equal access to digital technology for people with disabilities. Sally is her organization's representative on W3C standards groups and is also sits on a number of groups at the British Standards Institute (BSI) that relate to standards for ICT, this includes the group responsible for BS8878 the Code of Practice for Web Accessibility. Sally currently manages the Web and Online team at RNIB who deliver consultancy on website and app accessibility. She has led the writing of RNIB's own internal app standard for accessibility.


 **Twitter:** @sallycain www.rnib.org.uk

Dean Churchill / AT&T

Dean works on secure design, development and testing of applications at AT&T. Over the past several years he has focused on driving security requirements in mobile applications, for consumer applications as well as internal AT&T mobile applications. He has been busy supporting AT&T's emerging Mobile Health and Digital Life product lines. He lives in the Seattle area and enjoys downhill skiing and fly fishing.


John Gambrell

John has been in software development for over 20 years in corporate and consulting positions and has worked with paradigm shifts of client/server, web, and now mobile. He has been developing iOS applications since 2009 as a freelance developer and also does personal projects. He lives in Dallas, TX with his wife, kids, and three dogs.

 **Twitter:** @jpgdallas

Julian Harty / Commercetest

Julian was hired by Google in 2006 as their first Test Engineer in Europe, responsible for testing Google's mobile applications. He helped others, inside and outside Google, to learn how to do likewise; and he ended up writing the first book on the topic. He subsequently worked for eBay where his mission was to revamp testing globally. Currently he is working independently, writing mobile apps & suitable test automation tools, and helping others to improve their mobile apps. He is also writing a new book on testing and test automation for mobile apps.

 **Twitter:** @julianharty

Oscar Clark / Unity Technologies


Oscar Clark is an author, consultant and evangelist for Everyplay from Unity Technologies. He has been a pioneer in online, mobile and console social games services since 1998. He provided 'vision' for one of the first Online games communities (Wireplay - British Telecom); was global lead for games at Hutchison Whampoa (3UK) which included (perhaps) the first mobile in-App purchase; and was Home Architect for PlayStation®Home.

He is a regular columnist on PocketGamer.Biz and Develop-Online, an outspoken speaker at countless games conferences, a mentor for accelerator GameFounders and has guest lectured for several universities. His first book, "Games As A Service - How Free To Play Design Can Make Better Games" is available online.

 **Twitter:** [@athanateus](https://twitter.com/athanateus) www.gamesasaservice.net

Ovidiu Iliescu / Enough Software

After developing desktop and web-based applications for several years, Ovidiu decided mobile software was more to his liking. He is involved in Java ME and Blackberry development for Enough Software since 2009. He gets excited by anything related to efficient coding, algorithms and computer graphics.

 **Twitter:** [@ovvyblabla](https://twitter.com/ovvyblabla) www.oviduiiliescu.com
www.enough.de

Alex Jonsson / Evothings

Alex likes anything mobile, both apps and web technologies, and especially cleverly connecting physical stuff to mobile. He holds a PhD in CS/Media Technology from the Royal Institute of Technology in Stockholm and freely shares his ideas and thought with both the industry and academia. Dr Jonsson also has an eclectic urge to investigate how apps and services act as drivers for new business, by bringing novel values and ways to make things more connected, thereby binding the universe together in new, clever ways. Alex is founder and VP Community of Evothings simply because things are better when connected.



Twitter: @dr_alexj

www.evothings.com

Michael Koch / Enough Software

Michael has developed software since 1988 and joined the development team at Enough Software in 2005. He holds the position of CTO. He has led numerous mobile app development projects (mainly for Java ME, Android, Windows Mobile and BlackBerry) and he is also an expert in server technology. Michael is an open source enthusiast involved in many free projects, such as GNU classpath.



Twitter: @linux_penguin

www.enough.de

Daniel Kranz / Joule

Daniel is a multi-channel strategist with consultancy, agency and tech background. Previously a technical project manager at one of the leading advertising agencies and a mobile solution consultant for a mobile and multi-channel web specialist, he now works in global strategic planning advising brands on how to integrate mobile as part of their overall strategy.

www.jouleww.com

Carlo Longino / WIP

Carlo has more than a decade of experience in the mobile industry, beginning just after the turn of the century when he worked for Nokia at its headquarters in Finland. Before joining the Wireless Industry Partnership (WIP) as director of developer marketing services in 2010, Carlo worked as a freelance consultant and writer while he completed an MBA.



Prior to that, he was senior analyst for Floor64, a Silicon Valley-based analyst firm, where he covered the mobile and fixed telecom industries. He also helped launch and spent five years running TheFeature.com, a thought-leadership site owned by Nokia. Carlo has also been published in The Wall Street Journal, Business 2.0 and Dow Jones Newswires and has spoken at a number of industry events, including Mobile World Congress, SXSW, MobileBeat and CTIA, among others.

 **Twitter:** @caaarlo **www.wip.org**

Tim Messerschmidt / PayPal

Tim has been developing Android applications since 2008. After studying business informatics, he joined the Berlin-based Neofonie Mobile as Mobile Software Developer in 2011 and has consulted for Samsung Germany as Developer Advocate for Android and bada since 2010. In 2012 he moved to PayPal as a Developer Evangelist. He is passionate about Mobile Payments, UI, UX and Android development in general. Furthermore he loves to speak at conferences, writing articles and all kind of social media.

 **Twitter:** @seraandroid & @PayPalDev **www.timmesserschmidt.de**

Sebastian Meyer / D-LABS

Sebastian has more than a decade of experience with web and mobile technologies. He joined D-LABS as Software and Innovation Consultant after his studies in software engineering at Hasso Plattner Institute in Potsdam. Specializing in user-centered methodologies and innovation in an enterprise context, he consults with national and international startups, SMEs, and corporations.

www.d-labs.com

Patrick Mortara

Patrick studied computer sciences in Frankfurt. He has been developing desktop-based software since the mid-nineties, both as a freelancer and for his day job. He started mobile development in 2010 when Samsung released its first bada smartphone, the Samsung Wave I.

 **Twitter:** @pmortara **www.mortara.org**

Marcus Ross

Marcus is a freelance developer and trainer. After 10 years of being an employee in several companies, he is now doing SQL-BI Projects and everything mobile cross platform. He is a regular author in the German magazine "mobileWebDeveloper". In his spare time, he is often seen at conferences, speaking on mobile subjects and JavaScript. He also writes articles, books & tweets on mobile development.



Twitter: @zahlenhelfer

www.zahlenhelfer-consulting.de

André Schmidt / Enough Software

André has been in the software business since 2001. After starting his programmer's career in one of the leading companies of the defense industry, he joined Enough Software in 2007 as a mobile developer. In this position he created a wide range of mobile applications, mostly for Android. He is also a frequent speaker at developer conferences and bar camps.

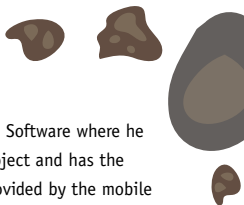
www.enough.de



Michel Shuqair / AppValley

Starting with black and white WAP applications, iMode and SMS games in the 1990's, Michel moved to lead the mobile social network Wauwee. Serving almost 1,000,000 members, Michel was supported by a team of Symbian, iPhone, BlackBerry and Android specialists at headquarters in Amsterdam. Wauwee was acquired by MobiLuck, which is now part of Paris based Madgic.com, a mobile monetization platform.

www.appvalley.nl



Marco Tabor / Enough Software

Marco is responsible for PR, sales and much more at Enough Software where he has worked for almost 7 years. He coordinates this book project and has the responsibility of finding sponsors and merging the input provided by the mobile community.



Twitter: @enoughmarco

www.mobiledevelopersguide.com

www.enough.de

Ian Thain / SAP

Ian is a Mobile Evangelist at SAP, though he started 13 years ago with Sybase Inc. He regularly addresses audiences all over the world providing mobile knowledge and experience for the Enterprise. He also writes articles, blogs & tweets on Enterprise Mobility and is passionate about the Developer & Mobile Experience in the Corporate/Business world.

 **Twitter:** @ithain scn.sap.com/blogs/ithain/ www.sap.com

Marc van 't Veer / Polteq

Marc is a mobile app test consultant and trainer at Polteq with over 8 years working as a coordinator and system tester. He has a lot of experience in testing in a technically oriented context, such as telecom, SOA, test automation, development of stubs and drivers and testing API's. In his current job Marc is part of a scrum team that develops the next generation of native apps for the Dutch and international parcel market.

marcvantveer.niobe.nl/blog www.polteq.com

Robert Virkus / Enough Software

Robert has been working in the mobile space since 1998. He experienced Java fragmentation first hand when developing and porting a mobile client on the Siemens SL42i, the first mass market phone with an embedded Java VM. After this experience he launched the Open Source J2ME Polish project in 2004. J2ME Polish helps developers overcome device fragmentation. He is the founder and CEO of Enough Software, the company behind J2ME Polish, many mobile apps, and this book.

 **Twitter:** @robert_virkus www.j2mepolish.org
www.enough.de

Chris Ward / Sitepoint

Chris is a globe-trotting developer and writer currently working on several projects that all aim to explore the potential of 'open culture'. Currently the editor of www.sitepoint.com/mobile he is always looking for new writers.

 **Twitter:** @ChrisChinch chrischinchilla.com

an initiative by



sponsored by



Microsoft

msdn.microsoft.com



developers.sap.com



hp.com/go/mobile

„If you want to start your own business in the mobile environment, this is the book you must read first.“

Minh Nguyen on amazon.com

„The title is grand, but the content does live up to the title to an extent. This ebook is a wide-ranging look at almost anything related to mobile app development out there.“

Azzief Khaliq on hongkiat.com

„A spectacular piece of work! You will be astonished by how incredibly fast you can establish your presence in the mobile market with the simple steps explained in this guide.“

Daniel Hudson on webtechman.com

„A must-read for everyone considering creating and marketing apps.“

Steve on amazon.com

Distribution Partner:



www.mobiledevelopersguide.com