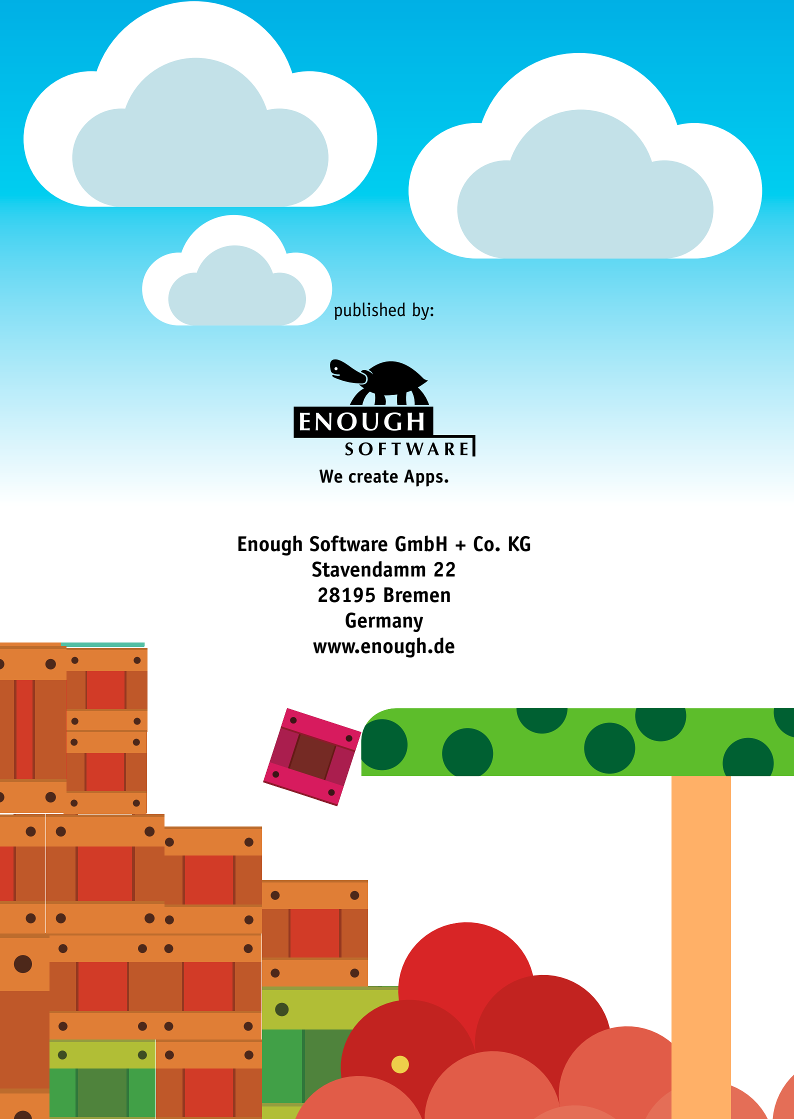
Don’t Panic

MOBILE DEVELOPER’S GUIDE TO THE GALAXY



published by:

ENOUGH SOFTWARE

We create Apps.

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

16th Edition February 2016

Please send your feedback,
questions or sponsorship requests to:
mobiledevguide@enough.de
Follow us on Twitter: *@MobileDevGuide*

Art Direction and Design by

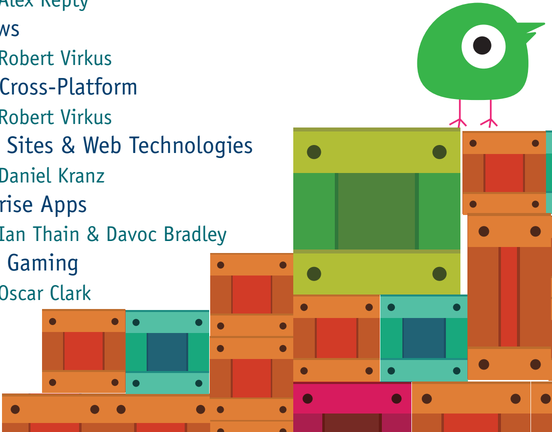
Cornelius Kwietniak
Mladenka Vrdoljak

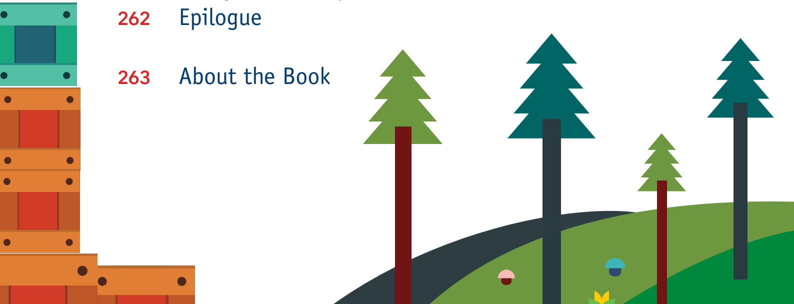
Editors:

Marco Tabor
Mladenka Vrdoljak

www.mobiledevelopersguide.com

Mobile Developer's Guide
Contents





Prologue

Another year, another big round of changes: Mozilla stopped Firefox OS and Jolla went through financial crisis that almost put an end to the project. BlackBerry released its first Android device (but promised to continue its support of BlackBerry 10). Google is now a part of their own family company “Alphabet”. Apple released Swift as Open Source and Microsoft released its “operating system as a service” Windows 10. So the mobile universe keeps evolving - and starts merging with wearable sector.

Due to these changes we decided to say goodbye to the dedicated Firefox OS, Java ME, BlackBerry 10 and Tizen chapters in this book and to focus even more on strategic and conceptual topics. One of the main progresses of the mobile industry in the younger past is that the user got more and more into the focus. Developers and app providers have understood that user feedback is a key factor of success. So we have extended the user-related content of this book even further with this edition: The concept and the design chapters are outlining how to involve users early and a completely new chapter about user feedback explains how to deal with your end-customers once you have gone public.

We would like to thank our dedicated authors - you all rock!

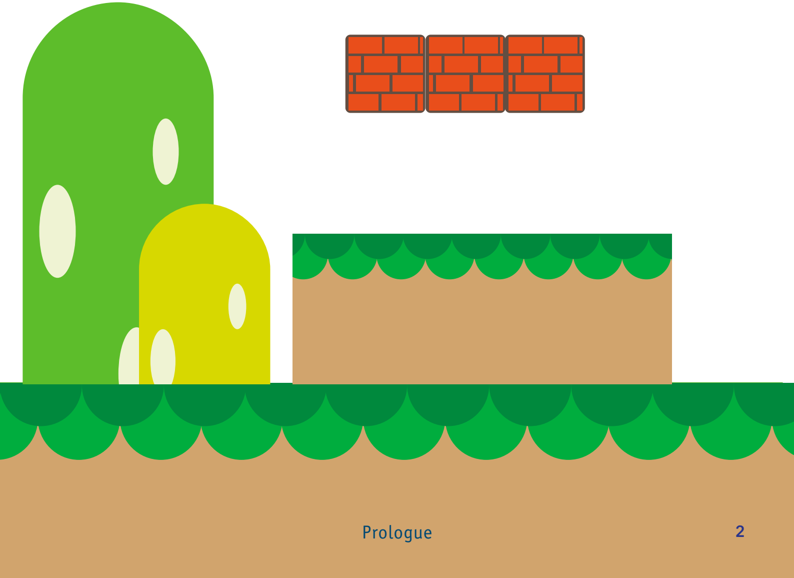
Another big thanks goes to our printing sponsor Microsoft - please visit their developer offerings at *dev.windows.com* and *azure.com*.

And now go ahead: Dive into the mobile universe and let us help you to find your very own place in it!

Robert + Marco / Enough Software

Bremen, February 2016

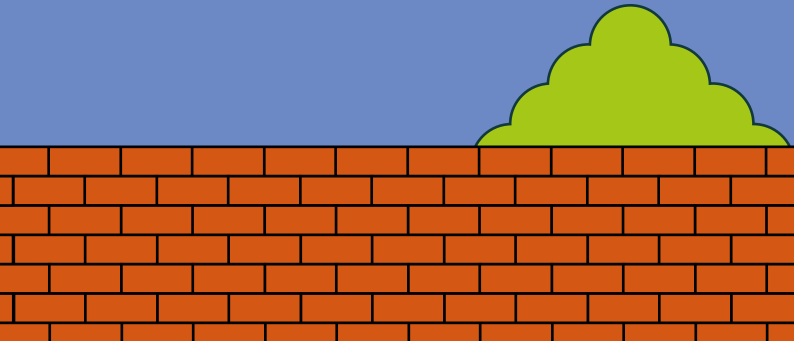
PS: Please follow us on Twitter *@MobileDevGuide* and visit *mobiledevelopersguide.com* to obtain the electronic edition of this booklet, which is available in several languages.



MOBILE DEVELOPER'S GUIDE TO THE GALAXY

1 PLAYER GAME

2 PLAYER GAME



BY Robert Virkus & Marco Tabor

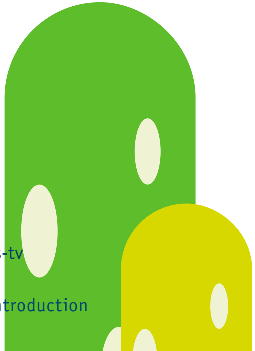
The Galaxy of Mobile: An Introduction

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; mobile has become the predominant form of computing in many areas already. Time spent on mobile app usage even surpassed the good old TV[1]-at least in the US.

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



1 dminc.com/blog/mobile-app-usage-surpasses-tv

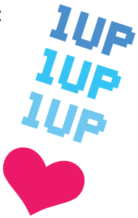
Topology: Form Factors and Use Patterns

Traditionally we app developers only targeted phones. Then tablets followed, and today our apps can span across a complete range of device types: smartwatch, smart glass, 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 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 realise a mobile service:



— App
— Website
— SMS, USSD[2] and STK[3]

App

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

2 en.wikipedia.org/wiki/USSD

3 en.wikipedia.org/wiki/SIM_Application_Toolkit

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.

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[4], 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 Phonegap[5].

Hybrid Apps

For many mobile app developers 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

4 apps.ft.com/home/web-app

5 www.phonegap.com

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.

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 realised 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.

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 which is transforming life and financial transactions in Kenya and other countries.[6]

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 chose 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 markets[7] (see the table below for global numbers). The Developer Economics Q3 2015 research[8] also

6 mpesa.in

7 www.theverge.com/2015/8/20/9181269/gartner-q2-2015-smartphone-sales

8 DeveloperEconomics.com

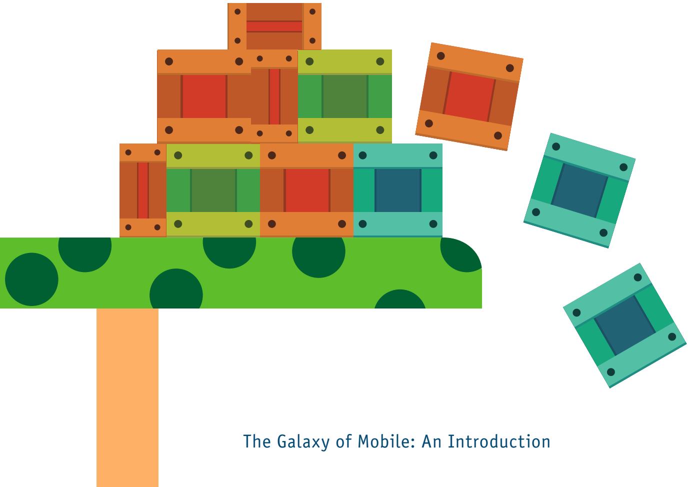
shows that iOS and Android are at the top in terms of developer mindshare - that is, the percentage of developers using each platform, irrespective of which platform they consider to be their 'primary'. Android was at the top, with 71% of developers currently working on the platform, followed by iOS with 51%.

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

Magnetar: Windows

While Windows Phone has had some successes, its world wide market share remained low - too low for many app developers. So Microsoft changed the rules with Windows 10 - now you can develop the very same app for both PC and Mobile (and for IoT, HoloLens and Xbox, too). So far that strategy seems to pay off as several major players have joined the Windows ecosystem lately.

To learn more about the platform and how to get started, check the respective chapter in this book.



Newborn Star: Ubuntu

Ubuntu for Phones powered devices have been entering the market in 2015. With its innovative concept and open source nature it has raised some interests in developer communities, but this seems to have little influence on sales volume so far. One of perceived main problems is the performance problems even on the higher end hardware like the Meizu MX4.

Your main point of entry is *ubuntu.com/phone/developers*. You develop your apps either as web apps or natively using Qt/C++. An interesting concept is scopes which allows users to view information without needing to launch your app explicitly. Frameworks are announced that should allow you to participate at and integrate into core activities like messaging.

Newborn Star: Tizen

Tizen[9] has enjoyed quite a success in the smartwatch market (compare our chapter about wearables), however previously promised mobile phones have been delayed by Samsung. In January 2015 the first Tizen powered smartphone, the Z1 has finally been launched in India. Seemingly gently yet continuously pushed forward by Samsung and Intel, Tizen aims to power not only smartwatches and smartphones but also TVs, tablets, netbooks and in-vehicle infotainment systems.

Typical Tizen apps are web based, but you can also create native C-based apps. Start your Tizen journey on *developer.tizen.org* and *developer.samsung.com/gear*.

9 tizen.org

Dark Star: BlackBerry 10

End of 2015 BlackBerry introduced its first Android handset, the BlackBerry Priv. At the same time BlackBerry promised to keep on supporting BlackBerry 10[10]. But the question remains if BlackBerry can keep up supporting two platforms at the same time.

You can develop BlackBerry 10 apps with Qt/C++, web technologies or - thanks to its Android compatible runtime - even as Android apps. Start your BlackBerry journey on *developer.blackberry.com*.

Super Nova: Sailfish OS

Will it explode? Jolla[11] - the company behind the Sailfish OS[12] - entered a debt restructuring[13] and laid off[14] about half of its employees. While future prospects may look dim, it is not over yet: Another financing round saved the OS for now but they had to give up the plans to enter mass production with their own tablet[15].

Start developing for Sailfish OS by visiting *sailfishos.org/develop*. One easy path to Sailfish is to use your existing Android app, but you can also create native Apps using Qt/C++ or even Python.

10 devblog.blackberry.com/2015/10/an-update-for-blackberry-10-developers

11 jolla.com

12 sailfishos.org

13 reviewjolla.blogspot.de/2015/11/news-jolla-financial-trouble-debt.html

14 reviewjolla.blogspot.de/2015/11/news-jolla-financial-difficulties.html

15 blog.jolla.com/new-years-greetings-jolla

Dark Matter: Feature Phone Platforms

While smartphones generally get the most news coverage, in some parts of the world feature phones are still pretty relevant. Even on a global level 22% of all phones sold have still been feature phones in Q1 2015[16], 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.

The big players in the feature phone market also had to realise this: Nokia shut down their feature phone app store in 2015.

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.

White Dwarfs: Firefox OS, 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. The latest one in that series is Firefox OS[17]. While we even had a dedicated chapter on that platform in our last Mobile Developer's Guide, Mozilla announced in December 2015 that development of Firefox OS on mobile phones will be discontinued[18].

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, we will see if that trend continues on the wearable front, too.

16 counterpointresearch.com/marketmonitor2015q1

17 mozilla.org/firefox/os

18 techcrunch.com/2015/12/08/mozilla-will-stop-developing-and-selling-firefox-os-smartphones

Solar System: Smartphone OS Market Shares

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

Platform	Market Share Q3			
	2015	2014	2013	2012
Android	84.7%	84.4%	81.2%	74.9%
iOS (Apple)	13.1%	11.7%	12.8%	14.4%
Windows Phone	1.7%	2.9%	3.6%	2.0%
BlackBerry	0.3%	0.5%	1.7%	4.1%
Other	0.3%	0.6%	0.6%	4.5%

(Source: *www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/*)

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 phones today[19], feature phones still outsell smartphones in some regions - and even grow in popularity, for example in Japan[20].

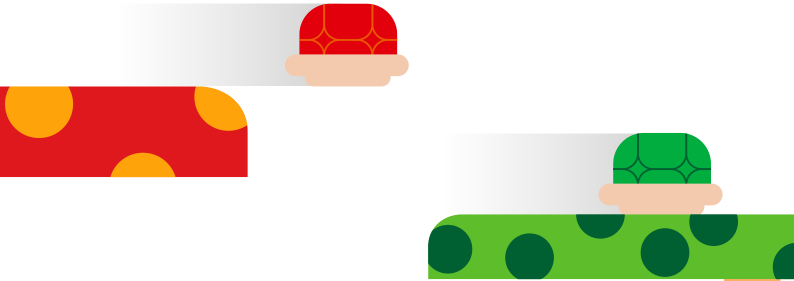
Also, be aware these are global figures: the regional market share of each platform varies significantly. In a world where localized content is increasing in importance, it is vital to know the details and characteristics of your target market. For example, China is the largest smartphone market today, but

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

20 reuters.com/article/japan-tech-mobilephone-idUSL4N0VM1HT20150216

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. At the same time, Apple is especially strong in the U.S.: In October 2015, 43.3 percent of U.S. smartphone subscribers were using an iOS device[21].

To find out about market share in your target region, check out online resources such as comscore[22], StatCounter[23], VisionMobile[24], Gartner[25], Statista[26] or Kantar Mobile World Panel[27].



21 Nevertheless, Android is leading in the US as well holding 52.9 percent of the market, see statista.com/statistics/266572/market-share-held-by-smartphone-platforms-in-the-united-states

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

23 gs.statcounter.com

24 visionmobile.com

25 gartner.com

26 statista.com/markets/418

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

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 FairPhone[28], ShiftPhone[29] or the GuardianProject[30] may prove more important than the OS or vendor of your choice in the future.

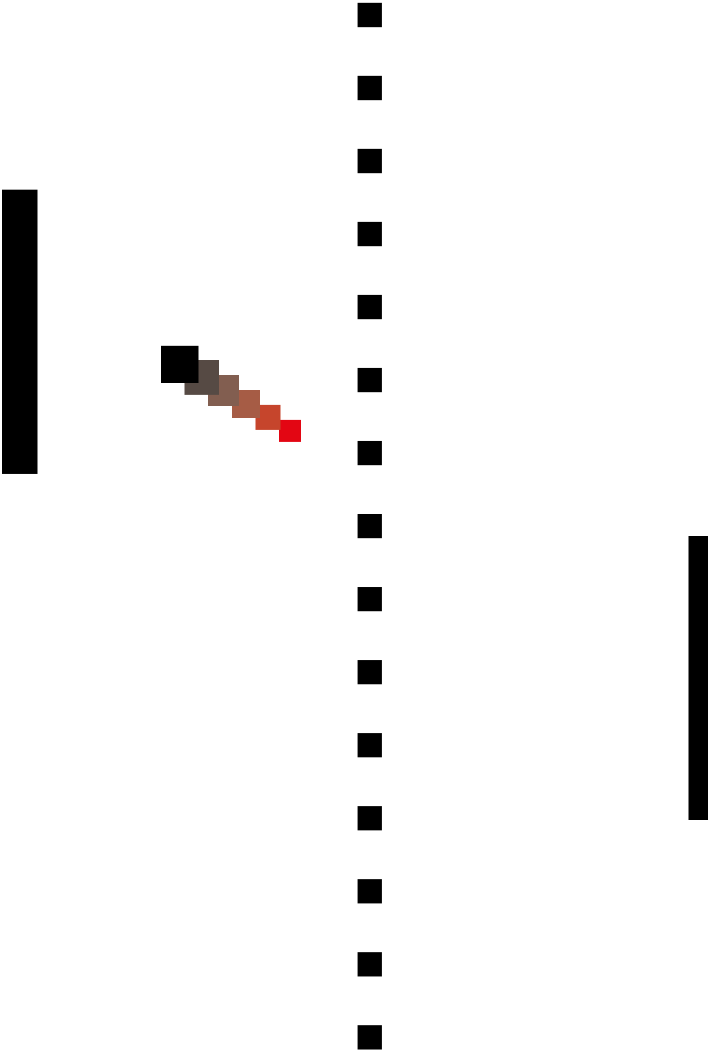
28 fairphone.com

29 shiftphones.com

30 guardianproject.info

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!



BY Sebastian Meyer

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. But please: Always consider the possibility that an app might not make sense for your project and that it might be the best idea to invest your money elsewhere. Otherwise you might end up sharing the experience of Birdly who summarise their learnings in an article entitled "Why you shouldn't bother creating a mobile app."[1].

The Key Elements of Success: Desirability, Viability & Feasibility

Let us 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 and value for them? A good product can simplify tasks or make people's lives more comfortable. 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

1 medium.com/inside-birdly/why-you-shouldn-t-bother-creating-a-mobile-app-328af62fe0e5#.9r624514i

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 could not 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. 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 backlash[2]. Tools like the Osterwalder Business Model Canvas[3] can help build your business case in a structured way. See the monetisation 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,

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

3 alexosterwalder.com

viability, and feasibility) to develop a fairly detailed concept before spending unnecessary efforts in implementing a solution. Get early feedback from others, including potential users to help refine your idea and concept.

Define the User's Needs

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 unorganised 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 built in.

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. Analysing 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.

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, but please do not rely on it as the primary source of information!

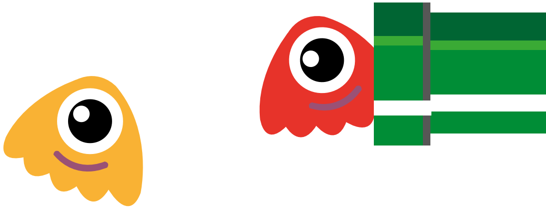
Furthermore, during requirements analysis, it is important to consider the differences between wishes and needs. According to Merriam Webster, a wish is "a desire for something to happen or be done."[4] Additionally, it is "an act of thinking about something that you want and hoping that you will get it or that it will happen in some magical way." This desire is normally conscious. Wishes focus on concrete material objects (i.e. smartphones) or on abilities (i.e. creativity). This is why users can express their wishes for a special product, like special features or colours. 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

4 merriam-webster.com/dictionary/wish

happy."[5] 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 eliminate the lack. Different lacks induce varying degrees of actions. According to the famous Maslow's need pyramid[6], 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.



5 merriam-webster.com/dictionary/need

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

Ideating

The first result of an 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 the ideation phase, it is important to start thinking very divergently and come up with a huge amount of solution ideas. If you create a large amount of ideas, it is more likely that you think "outside the box" and that really innovative ideas to address the stated problem come up.

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 method[7], visual confrontation, or the Disney method[8]. 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 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 it reduces the risk that ideas are evaluated too early and that less dominant persons are not heard.

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

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

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 really understands them the same way as you. Natural language is often interpreted differently and creates different mental models for each person. Thus, understandings 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 visualisation 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 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.

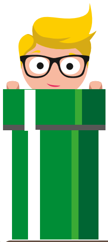
When it comes to software engineering, there are three artifacts related to the term prototype: wireframe, mock-up, and proof-of-concept implementation. Wireframes and mock-ups are usually used in early phases to validate a concept, information architecture, and basic interaction. Proof-of-concept implementations 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 app prototypes.

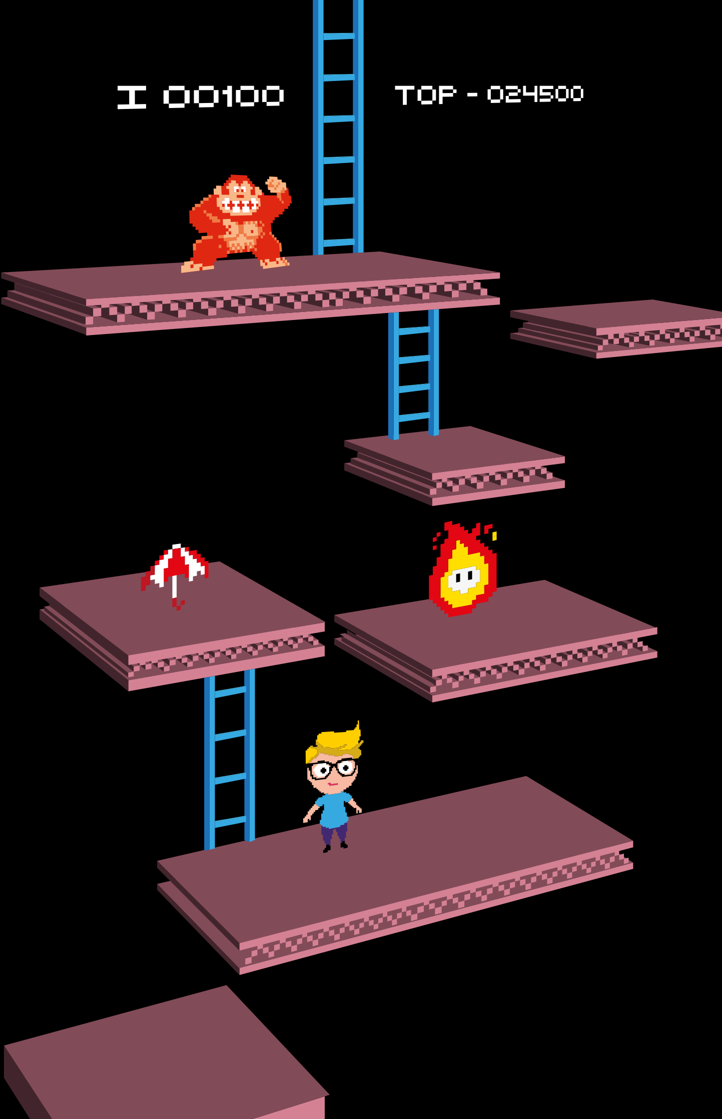
Learn More

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 book[9], 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.
- *uxbooth.com/articles/complete-beginners-guide-to-design-research*, the "Complete Beginner's Guide to Design Research" by Andrew Maier. The title is a little exaggerated, but it is a good introduction with useful further links.
- *theleanstartup.com* the website of the Lean Startup book by Eric Reis. The Lean Startup helps readers discover ways to create more desirable products with less waste.

9 businessmodelgeneration.com/book




I 00100
TOP - 024500

User Experience & User Interface Design

BY Anna Alfut

At this point you probably have a concept for your app. In the previous chapter you learned how to generate and validate multiple ideas. This section is about the methods that are useful for transforming your concepts into usable products through iterative design. It focuses on describing a range of techniques to shape your vision. You can use them to outline early ideas at a high level, as well as define detailed guidelines for the final implementation.

User Experience (UX) is how your proposition is generally perceived by the end users. Was the journey easy and clean? Did they have enough information at each step? etc. It is a broader term that can be applied to designing services and software products. User Interface (UI) design is a term that is specific to software products. Whether it is a website or a stand-alone app, the UI is where your users will experience what you offer to them through interactions, visual design, flow, messaging etc.

Designing an application from scratch requires a different approach than improving on an existing product. In both cases you will frequently switch between the big-picture and detailed design mindsets to make sure that you continue to improve quality while staying on track to complete your vision.

You should use the methods described in this chapter 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. If you are on a very early stage and just need some way of outlining your concept, you can use sketching or drawing a user journey map. When you get to another design iteration and need to refine UI the same methods can be used just with different attention to detail.

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

User Experience (UX)

To give your app's users the experience they will appreciate you need to be very clear about the goals of your project. What is it that will be different about your app? What kind of people you are hoping to attract? etc. Methods described here are useful for capturing in a visual and logical way the larger context of your project.

Personas

Personas are fictional profiles that represent your application's audience.

The persona's profile structure usually contains information like name, age and a profession. A description of their specific interests and how a particular proposition fit in their lifestyle. How does it help them to achieve certain tasks? In other words, what would motivate these users to give your app a try?

Typically a project has more than one personas. Ideally the profiles should come from previous research, but even if they are based only on your guesses, writing them down is one of the most important steps you can take. For each persona you can select a profile picture to get some visual reference. It helps in building empathy towards the users this persona represents.

Why do you need them? Well, it is not uncommon for the makers of a product to approach problem solving based on their own experiences, preferences and previous knowledge about a certain domain. Acknowledging different user needs early sets you off to a good start to avoid this trap. Whichever step you take from here think of how your users will perceive the design if they saw it for the first time.

User stories

Once you get a better idea of the type of users you are designing for, you can start defining more granular scenarios - user stories. Thinking in user stories is thinking in flows - the journeys from point A to B within the application. An example of a story would be a purchase flow, uploading or sharing a photo, making a note etc.

By listing the 'must have' stories early on you will get an idea of how much work there is ahead before the application is ready to be released. This will allow you to capture the full experience before diving into the interface details. A list of stories is the brief for your app, it captures the main user needs, together with knowing your users's persona profile you can start now get on with the detailed design.

After you have the main journeys outlined, you can break each of them into smaller steps - tasks. Each task is a separate micro flow within a larger journey. For the sample flow of uploading a photo, think whether users will need to access a gallery on their phone? Do they need to select a picture? Can they add more than one at one? All of those activities contribute to the overarching story.

When you think about a solution to a particular problem keep coming back to your personas profiles. Some users will need more guidance than others and you might consider putting extra messaging in the UI for them. Even if it is unlikely that everyone will notice every bit of information design that you put in place, as long as it is available for those that need it, and balanced, so it does not get in the way of more independent users, it will work for the wider group.

Flow diagram

A Flow Diagram demonstrates steps in the flow and the major decisions points users will run into while navigating inside your app. It is more logically defined and allows you to think through the possible journey's routes in a visual way.

You can build up a flow chart for the whole application or 'zoom in' and focus on a particular journey. On a higher level a flow diagram outlines the app structure. For example, you can identify how many different screens you will need to design. When you focus on a specific journey it is a great technique to capture all cases that you need to design for (like, have you thought about what to show to users if there is no internet connection available?). As your project mature you can iterate on your diagrams and update them to reflect the accurate state of your design.

You can read more about the chart's building blocks on Wikipedia[1].

1 en.wikipedia.org/wiki/Flowchart

Experience map

As the name suggests, an experience map is an overall impression of the entire experience you are creating for your users. This kind of overview usually takes form of an infographic/poster that others involved in your project can understand and relate their contribution to. To illustrate the steps in user experience you can use the screen shots from your app or, if it is early days, or you want to keep the map more implementation independent, go with illustration or simply text.

Depending on the idea your app can contain the entire user experience or can be only one of the touch points on the map alongside the customer service centre, a supplier, delivery service, website, social media channels or anything else that fits into a setup of your specific idea.

If your application is only one part of a wider experience, drawing the map is an excellent exercise to understand and demonstrate the bigger context in which the application will sit in. When you move to the UI definition and interaction design, having those the touch points crossovers identified is really useful. Understanding those dependencies will inform how you create an experience of sending users away from your app or welcoming them back again. The map details can vary for different personas and can also change depending on whether it shows an initial - first time experience or a return journey.

User Interface (UI)

When you understand how individual journeys fit together in the bigger picture you can move into more defined UI design. Techniques described in this part are useful to define design in enough detail that it will resemble the end product. Even a sketchy version of the UI will be instantly recognised by users as a representation of the interface, which will allow you to ask for feedback early.

This dynamic of going from big picture thinking to details and back again is in itself a great illustration of the iterative nature of software design. With each cycle you get more confident that the overall concept makes sense (or if it does not, you can adjust the course as and when needed) and you can move into applying next layer of detail until all parts are in place.

Sketches, wireframes and mockups

To convey various UI ideas you can simply sketch on paper, or you can use one of the many available digital tools to draw a wireframe. Wireframes are low fidelity 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. Those basic compositions visualise how complex the layouts and interactions of your app's screens will be.

Frequently updated wireframes are an excellent reference point for discussing details and next steps with everyone on the team. And if you work on your own, they help to organise your own workflow. If you need to identify UI pattern libraries at a later stage, those will be guided by repeatable elements from your sketches.

So how detailed should your wireframes be? Sketches are ideal at the early stage for quick ideas generation. The more specialised wireframing applications come with libraries of ready-made widgets to quickly arrange on a screen. The advantage of using digital tools is that you have an editable version of your screens 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 version is easier to update and maintain.

Mockups is a term commonly used for a wireframe with more visual design detail applied. Some mockups can look identical to the final implementation. The more refined visualisation of the final product can be useful for demos, sometimes a quick sketch is enough for a highly collaborative teams to agree on next steps. It is up to you to decide how much of the finished look you need on each step to make progress.

Messaging

As soon as you start using meaningful labels and titles, rather than the placeholder text, you start defining the way you communicate with your users. Depending on your application, the language will have a different role in guiding your users through the flows. But even if the use of words in your UI is minimal, do not leave it in a placeholder state too long.

The 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 work against what you are trying to achieve.

Prototypes

An interactive prototype is the best way to visualise and evaluate your app's interactions. It is usable enough to communicate the design, so you do not need to provide as much documentation as you would need to annotate static images. Your prototype can have visual design applied and look exactly as it will after the implementation, or you can stay on the wireframe level and focus on functionality and content.

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. Many prototyping tools allow you to experience your concept on an actual device. Take the advantage of it.

Prototypes are usually developed before you spend time on implementing code and pixel perfect design. 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.

In terms of putting a prototype together there is no single best solution. You can use whatever technique works for you. From paper prototyping to using one of the specialised tools or other applications that have the functionality to put clickable journeys together. If you have coding skills, building a HTML prototype is another good way to go. You can also rapidly prototype on the existing app, it all depends on what approach works for a specific project setup. In that sense everything can be seen as a prototype until it is released.

Prototyping & wire framing tools

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

- **Axure:** *axure.com*
- **Balsamiq mockups:** *balsamiq.com*
- **Framer:** *framerjs.com*

- **Mockingbird:** *gomockingbird.com*
- **OmniGraffle:** *www.omnigroup.com/omnigraffle*
- **Origami from Facebook:** *facebook.github.io/origami*
- **Penci:** *pencil.evolus.vn*
- **POP:** *popapp.in*
- **Principle:** *principleformac.com*
- **Proto.io:** *proto.io*
- **Sketch:** *sketchapp.com*

Interactions & animations

When you start working with a prototyping tool you can visualise the way users will interact with your app in a very direct way. By putting the prototype on a device you can actually experience how your app will feel. This experience is extremely valid not only in user testing, quite often you will be able to identify the missing elements for yourself.

The way your UI moves and respond to users actions will contribute to how usable your software is. 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 interactions. Following the recommended practices will make your app instantly easier to interact with for users that are already familiar with their device's patterns. For more information and links to specific online resources see the platform - related chapters of this guide.

With an ever-changing mobile devices market you should also consider how your UI will look on different screen sizes. 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 wireframing 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 the layout scaling problem for a while. Also check if the platform specific guidelines provide more information around this subject.

Visual design

Unless you are building an app that uses a non-visual input/ output, your app's 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.

You would have already applied a number of graphic design principles during wireframing stage. These include layout design elements like spacing and visual hierarchy. Polished visual design will not only improve your UI's aesthetic appeal, a well-executed branding enhance your app's functionality and reduces the learning curve for users by providing 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 colour from screen to screen, consider the impact on the users. Will they be confused? Will they understand the reason behind the change? If the style alterations are intentional make sure you are doing them for usability reasons.

Similar to designing layouts and interactions on the prototyping level, certain styling decisions might be informed by specific platform guidelines. Your app can look very different depending on which platform it was defined 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 your app. Use the background, controls colour and maybe certain images or layout choices to achieve the brand's look and feel. A splash screen (if present) is a 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.

Visual Design is a rich area and if you would like to understand more about specific techniques you can find more information and examples online - see below for some recommended websites.

User testing

The best way to validate your interface concept is to show it to users as soon as your work is representative enough to prompt 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. It will expose concepts that do not 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 that on a clickable prototype that you can update quickly.

Typical user testing session is about an hour long. 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 personas descriptions and look for users that match those profiles.

To make the best use of the testing time, prepare in advance. Note down introductions, think of how you will explain the session dynamic 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 out and have the notes handy to make sure that you do not forget to ask.

It is good to mention to users that the prototype is not complete and there might be some unfinished parts. If they assume that the person that is 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 the testing session was to get an 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 by revealing how things work (unless they cannot figure it out and you cannot proceed with the session) and word your questions in a non-interruptive way. During the session either record user feedback or make sure to take enough notes.

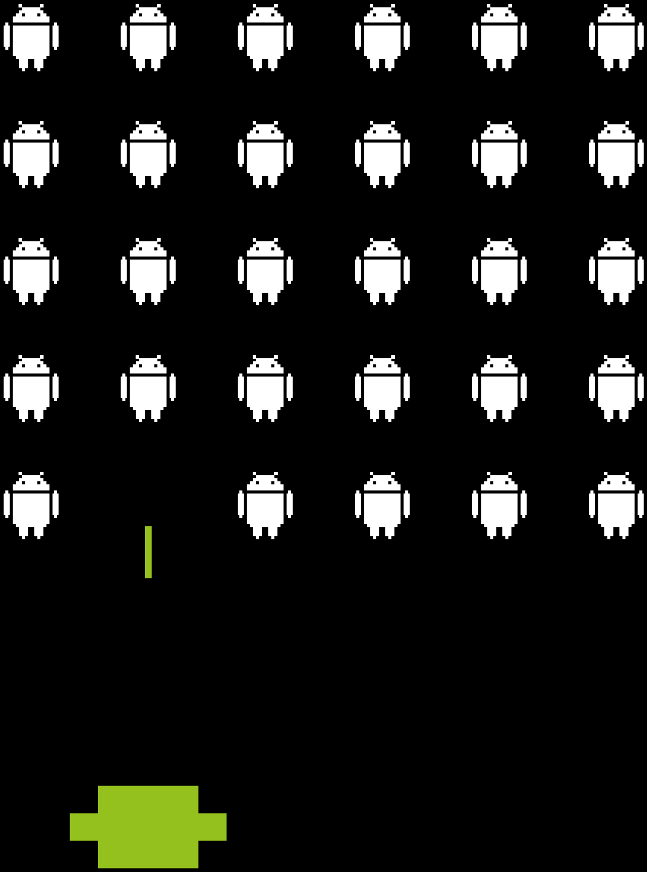
When you get feedback, you can reiterate your design and improve the parts that were not quite complete or if the feedback was good move on to the development phase.

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. A single non-biased opinion is better than no opinion at all. If you are still exploring new areas and your own prototype is not quite ready, you can run testing sessions on other apps that have been already released. It can surprise you how much others notice about the application that you might never have thought of.

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*



BY Vikram Kriplaney & André Schmidt

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 ever which today dominates the market: More than 82% of all smartphones sold in Q2 2015 worldwide were based on Android[1], 71% of all mobile developers are targeting Android[2]. In September 2015, Google announced that over 1.4 billion Android devices have been activated so far[3] 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. The number of Android apps on Google Play has surpassed 1.8 million in December 2015[4].

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 months or so. The latest release is Android 6.0 'Marshmallow' which

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

2 sometimes among others, see the Developer Economics Report available via developereconomics.com

3 androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide

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

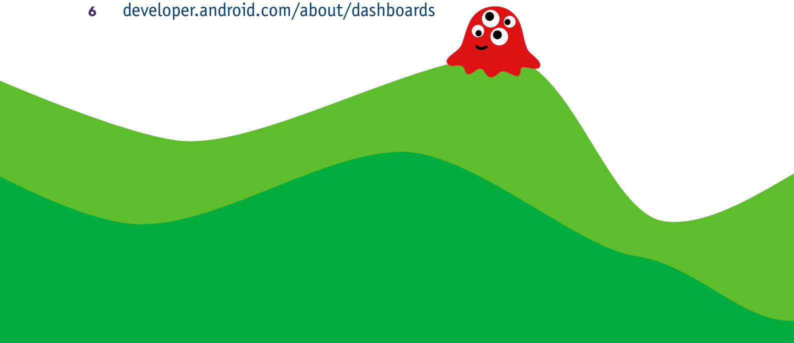
introduced new features such as a new permission and power management system and improves upon the major features of its predecessor L 'Lollipop'. The most major improvement of Lollipop was the introduction of a new UI toolkit called Material Design[5] which aims to unify all targeted platforms like phones, wearables and TVs with one design approach.

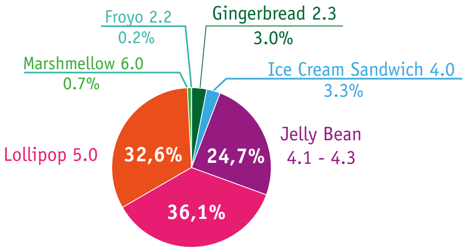
Other major Android L enhancements include the new Android Runtime (ART) that provides features like a smarter garbage collection that ultimately improves performance by a factor of up to 4x. Project Volta introduced tools to improve the battery life by optimising the behaviour of an app. Further features 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 above[6].

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

6 developer.android.com/about/dashboards





(Source: *developer.android.com/about/dashboards, data from January 2016*)

Android Marshmallow

Android 6.0 "Marshmallow" is the most recent version of Android OS, announced at the Google I/O conference in May 2015 under the codename "Android 'M'". It has been officially released in October 2015.

One of the more significant changes in Marshmallow is the new permissions architecture. Users no longer have to grant all the permissions that an app requires during installation. The app can (and should) now request permissions as and when they are required. Developers are encouraged to follow the current best practice of asking for permission just when it is needed, giving the user as much context as possible.

Marshmallow also introduces new APIs for contextual assistants (like "Google Now On Tap"), a new power management system that reduces background activity when a device is not being physically handled, native support for fingerprint recognition and USB Type-C connectors, and the ability to move data to removable storage (e.g. a microSD card).

Material Design

During the Google I/O conference in June 2014, Google unveiled their new design language based on paper and ink, named Material Design. Originally codenamed Quantum Paper, Material Design extends the "card" concepts first seen on Google Now. Says designer Matías Duarte: "unlike real paper, our digital material can expand and reform intelligently. Material has physical surfaces and edges. Seams and shadows provide meaning about what you can touch."

Perhaps for the first time, Material Design brings a strong, consistent visual identity to the Android ecosystem, parallel but distinct from iOS's flat design and Windows' Metro design. The visual language itself is well specified and documented.[7]. To encourage a solid user experience and consistent appearance of Android apps, Google provides a design guide[8]. Going into the importance of colour schemes, design patterns and the new Material design, the guide provides a great orientation when building apps for the Android ecosystem.

There is also a canonical implementation of Material Design for web application user interfaces called Polymer Paper Elements[9].

Android Wear

Android Wear[10] was announced by Google in March 2014, in partnership with Motorola, Samsung, LG, HTC and Asus. It is basically the Android OS ported to smartwatches and other wearable devices, which can pair with phones running Android version 4.3 or newer (there is some limited support for pairing

7 www.google.com/design/spec/material-design

8 developer.android.com/design

9 elements.polymer-project.org

10 developer.android.com/training/building-wearables.html

with an iPhone). Wear devices integrate Google Now and the Google Play Store.

A key feature is the Google Fit ecosystem of apps that support run and ride tracking, heart activity, step-counting, etc. Users can use their watch to control their phone – music, for example. Notifications via the vibration engine are another key element. Those can be used for notifications from Google Now like flight reminders, traffic warnings, meeting reminders, etc.

Android TV

Announced at the Google I/O conference in June 2014, Android TV[11] was based on Android 5.0 "Lollipop" and is a successor to Google's previous smart TV initiative, Google TV. Android TV is designed to be built into TVs as well as standalone digital media players. Google developed the first Android TV device with Asus – the Nexus Player[12], released in November 2014. Several TV manufacturers, including Sony, Sharp and Philips, have integrated Android TV into their screens today. Users can download apps and games from the integrated Play Store, including media apps like YouTube, Hulu and Netflix.

Android TV apps use the same structure as those for phones and tablets. Developers can thus leverage their existing apps and knowledge to target the TV platform. See *developer.android.com/tv* to learn how.

Getting Started

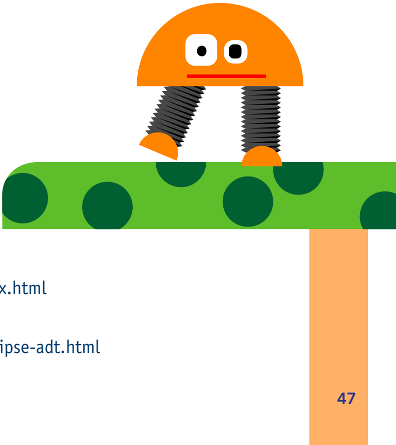
The main programming language for Android is based on Java. But beware, only a subset of the Java libraries and packages are supported and there are many platform specific APIs that

11 android.com/tv

12 www.google.com/nexus/player/

will not work with Android. You can find answers to your "What and Why" questions online in Android's Dev Guide[13] and your "How" questions in the reference documentation[14]. Furthermore, Google introduced a section in their documentation called "Android Training"[15] 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 optimising your app for better battery life.

To get started, you need the Android SDK[16], which is available for Windows, Mac OS X, and Linux. It contains the tools needed to build, test, debug and analyse apps. The Android Development Tools (ADT)[17] are responsible for the integration with IDEs and making sure that your development flow is as comfortable as possible.



13 developer.android.com/guide

14 developer.android.com/reference

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

16 developer.android.com/sdk

17 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. Android Studio[18] 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"[19] section.

Native development

The Android NDK[20] 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 its non-native pendants. This is important for game developers and anyone who needs to rely on efficient processing.

18 developer.android.com/sdk/index.html

19 developer.android.com/tools

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

Implementation

App Architecture

Android apps usually include a mix of Activities, Services, BroadcastReceivers and ContentProviders; these all need to be declared in the application's manifest. The manifest also includes the metadata of an application, like the title, version and its required permissions.

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 BroadcastReceiver handles messages broadcast by the system, your own or other apps. A ContentProvider 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 behaviours 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, localised strings and images are organised 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 analyse method trace logs, inspect layouts and test apps with random events, can be found in the tools directory of the SDK. If you are facing issues, such as exceptions being thrown, be sure to check the ddms log or use the logcat mechanism.

If you are using features such as Fragments[21] for large screens, be sure to add the Android Compatibility package from Google. It is available through the SDK and AVD Manager and enables development for older Android Versions using modern features. 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 online[22].

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 framework[23] which gets updated via the Play Store

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

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

23 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 list[24].

To automate testing, the Android SDK comes with some capable and useful testing instrumentation[25] 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. MonkeyRunner[26] 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

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

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

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

efficient UI automation testing framework[27] 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`.

Espresso[28] provides a very lean API that helps to quickly write procedural tests for your UI.

Open source testing frameworks, such as Robotium[29], can complement your other automated tests. Robotium can even be used to test binary apk files if the app's source is not available. Roboelectric[30] 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 respective JVM. Plugins such as the Android Emulator Plugin[31] enable these tests to be configured and run in Hudson and Jenkins.

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

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

29 code.google.com/p/robotium

30 robolectric.org/

31 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 flexible ways to build Android apps. Gradle[32] is now the officially supported build automation tool for Android. There is also a maven plugin[33] which is well supported by the community. Both tools can use dependencies from different Maven repositories, for example the Maven Central Repository[34].

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"[35].

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.

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

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

34 www.maven.org

35 gradleplease.appspot.com

Distribution

After you have created the next killer application and tested it, you should upload it to Android's appstore called "Play" at *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 localisation service, you can easily add new languages to your app by paying a fee - make sure to check the Localisation Checklist[36] for detailed information about the importance of this topic.

As there are lots of competing applications in Android Play, you might want to use alternative application stores[37]. 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.

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

37 onepf.org/appstores/

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 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.

Additional to the versions of the major manufacturers, the Android Open Source Project (AOSP)[38] offers an open source version of the Android stack to create custom ROMs and port devices to the Android Platform. Independent manufactures like Fairphone[39] use AOSP to create their version of the Android platform. The downside of this approach are missing Google services like the Google Play Store as normally available on mainstream Android devices.

38 source.android.com

39 fairphone.com

Monetisation

In addition to selling an app in one of the many app stores available, there are several different ways of monetising 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 service[40] that utilises the Google Play Store or PayPal's Mobile SDK[41] and Mobile Payments Library[42]. Most services differ in transaction-based fees and the possibilities they offer for example subscriptions, parallel payments or pre-approved payments. If you are looking to bring extra cool functionality to your app, you should consider implementing card.io's SDK[43] 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 optimised monetisation.

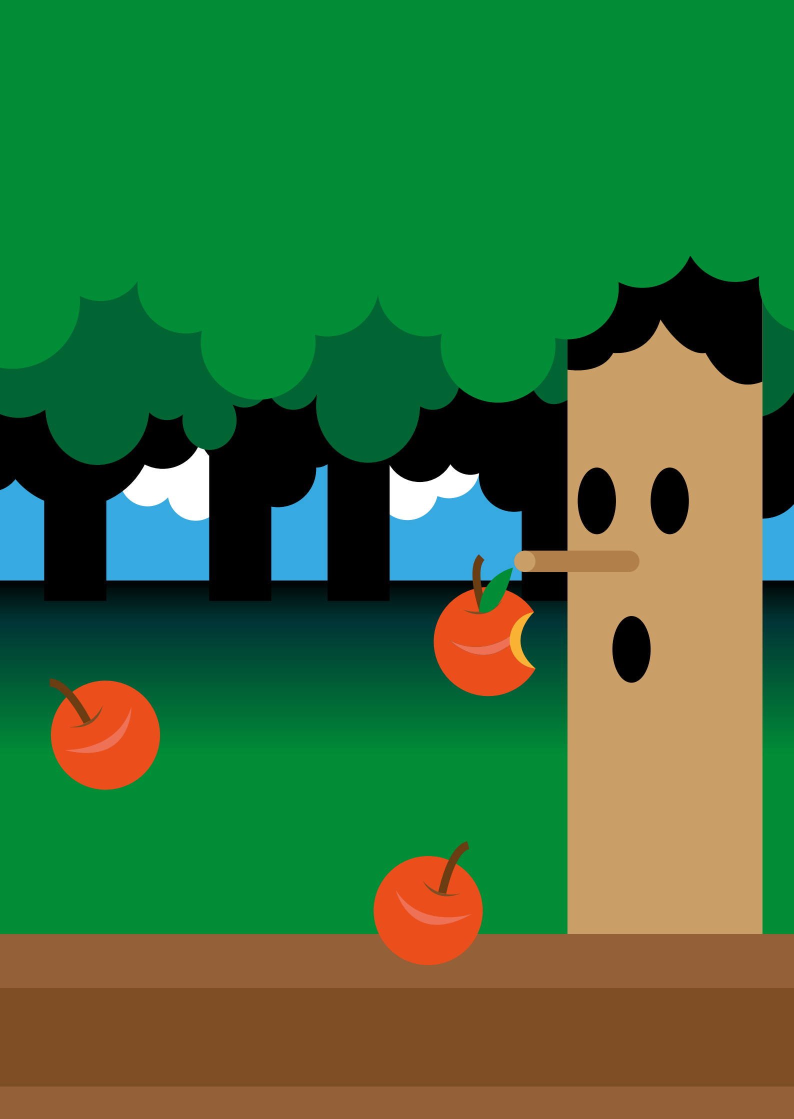
Be sure to check that the payment method of your choice is in harmony with the terms and conditions of the different markets you want to publish your app to. Those particularly for digital downloads, for which different rules exist, are worth checking out.

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

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

42 developer.paypal.com/webapps/developer/docs/classic/mobile/gs_MPL/

43 card.io



BY Alex Repty

iOS

On January 9th 2007 Steve Jobs unveiled a new product category for the then computer and music device company. 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"[1]. 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, iOS 9, that launched in September 2015.

Where iOS7 was touted as a major UI refresh the focus of the features in iOS8 were the new frameworks and services as well as the newest device, Apple Watch. It also introduced 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. iOS 9 then enhanced multi-tasking and introduced deep linking.

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

The Ecosystem Today

Developing for iOS is more popular than ever. In Q3 2015, the number of iOS apps surpassed 1.5 million[2]. Total app revenues on iOS in 2015 alone summed up to $20 billion, which means that iOS developers have earned a cumulative $40 billion from Apple's AppStore[3].

iOS Install Base

In addition to selling over one billion 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. Twelve weeks after the launch of iOS 9 Mixpanel already reported an adoption rate of nearly 80% of all iOS devices[4] with 16% still on iOS 8, leaving only 4% of devices running an older iOS version. Contrast this with Android's OS version 5.0/5.1 Lollipop, which after a year has not managed to capture even a third of Android devices[5]. The latest release, 6.0 Marshmallow has only being used on 0.5% of Android devices two months after its general release.

2 statista.com/statistics/276623/number-of-apps-available-in-leading-app-storesapps

3 www.apple.com/pr/library/2016/01/06Record-Breaking-Holiday-Season-for-the-App-Store.html

4 mixpanel.com/trends/#report/ios_9

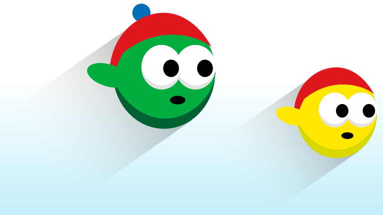
5 developer.android.com/about/dashboards

Devices Running iOS

Instead of listing every device Apple has created that runs iOS, here are the current devices that support iOS 9 as these would be what a new developer today should target:

- iPhone - 4S, 5, 5C, 5S, 6, 6 Plus, 6S, 6S Plus
- iPod Touch - 5th & 6th generation
- iPad - 2nd, 3rd & 4th generation, Air, Air 2, iPad Pro
- iPad Mini - 1st generation, Mini 2, Mini 3, Mini 4
- Apple TV (tvOS, a subset of iOS)
- Apple Watch (watchOS, a subset of iOS)

A detailed list of iOS devices, their capabilities and supported iOS versions can be found on Wikipedia[6].



6 en.wikipedia.org/wiki/List_of_iOS_devices

The Architecture

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 (see Figure 1). 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.

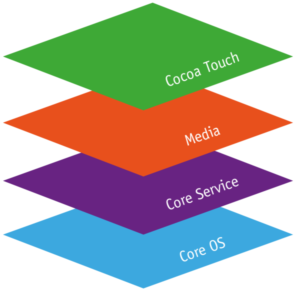


figure 1

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.

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 OS X. Version 3 of Xcode supported the first iPhone SDK in 2008 and the most recent version is Xcode 7, released with iOS 9 in September 2015. Xcode is an integrated development environment 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 iOS 9 supporting six different screen sizes. Instead of supporting all of them 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 is apparent that Apple prefers developers to take advantage of it. Something a new iOS developer should consider.

Objective C

Objective C has it 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 language[7]. 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 criticised 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# while providing improved legibility through named parameters and verbose class and method names. Incre-

7 en.wikipedia.org/wiki/Objective-C

mental 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 stated Swift is influenced by other languages such as C#, Ruby, Haskell, Python and countless others[8]

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.

8 nondot.org/sabre

In early December 2015, Apple open-sourced Swift[9] along with a bunch of related tools, frameworks and examples. Apple is actively engaging the community in the future development of the language by soliciting feedback, proposals for new language features and pull requests. Less than a week after it was first open-sourced, Swift is already the #1 open source programming language on GitHub[10], overtaking other popular languages like Ruby or PHP.

Performance Tools and Testing

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

Instruments allows developers to collect data about the performance and behaviour 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 is the test framework integrated with Xcode to provide extensive testing in an organised 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 analyse the results using the Test Navigator, all from inside Xcode.

9 github.com/apple/swift

10 github.com/showcases/programming-languages

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 10.10 (Yosemite) 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.

Distribution

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 tries to 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.

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** 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 organised 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 Guidelines[11]. It offers developers 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.
- **Swift.org**, the Swift community's official home
- **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

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

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 organised 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"[12] 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 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.
- **This Week in Swift** is a weekly newsletter involving the most interesting Swift-related news, developments, tutorials and general tidbits related to iOS development.

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

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 whereas 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 Azam[13] that lists several screencast tutorials for iOS.

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 www.youtube.com/user/azamsharp

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.

BY Robert Virkus

Windows

Thanks to the well received product and to Microsoft's aggressive marketing, Windows 10 has gained well over 110 million PC users by November 2015. With its "Universal Windows Platform" (UWP) you can target the widest range of devices:

- Mobile device family: Windows phones, phablets
- Desktop device family: Tablets, laptops, PCs
- Team device family: Surface hub
- IoT device family: Compact devices such as wearables or household appliances
- Living room: Xbox One
- Augmented reality device: HoloLens

Your main entry points as an app developer or app provider are *dev.windows.com* and *design.windows.com*.

The Ecosystem

While Windows still dominates the PC market, Windows 10 Mobile (formerly Windows Phone) market share remains small. According to Kantar World Panel[1], the market share fluctuates between around 10% in some European markets and Russia and 0.4% in Japan in the 3 months period ending in October 2015. The Surface brand got its recognition and apparently more Windows tablets than Apple iPads were sold in October 2015 according to 1010data[2].

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

2 winbeta.org/news/microsoft-beats-apple-online-tablet-sales

Thanks to the strong Windows 10 performance, Microsoft hopes to decrease the app gap which again makes its Windows 10 mobile offering more attractive. So far that strategy seems to work out with various major app players joining the Windows platform lately. According to the Developer Economics Report Q3 2015, 44% of mobile developers are planning to adopt Windows 10[3].

Languages and Tooling

As a Windows developer you can choose one of the following language and UI toolkit options:

- C# or Visual Basic with XAML
- JavaScript with HTML/CSS
- C++ with XAML
- C++ with DirectX

The main IDE is Microsoft Visual Studio IDE[4], which nowadays also supports Android and Crossplatform/Cordova development.

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 Win2D[5] or game libraries such as MonoGame[6]. Between the XAML UI stack and DirectX you can now also access the intermediate layer

3 developereconomics.com

4 visualstudio.com

5 github.com/Microsoft/Win2D

6 monogame.net

with the UI Composition API[7], which can be easily accessed from the .NET languages, too.

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 UWP XAML approach. While the UWP based apps are the future, you can still create Silverlight apps for Windows 10 Mobile devices.

Last but not least, there are several options to create apps without deep development skills:

- **PowerApps**[8] are aimed at businesses to connect various data sources and create apps for Android, iOS and Windows without coding skills. Backed by Azure services in the background.
- **Windows Phone App Studio**[9] allows you to create Windows apps with the help of templates.
- **Project Siena**[10] provides a WYIWYG environment for creating business themed Windows apps.

7 msdn.microsoft.com/library/windows/apps/mt592880

8 powerapps.microsoft.com

9 appstudio.windows.com

10 microsoft.com/projectsiena

Microsoft Design Language

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 Language[11] has been extended to the Xbox and Windows 8 and evolved into the Microsoft Design Language 2 or MDL2 for short. This UI paradigm contains the following principles:

— **Content not Chrome** removes unnecessary ornaments and lets the content itself be the primary 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 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.

Designers will find many inspirations and information at *design.windows.com*.

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.

11 wikipedia.org/wiki/Metro_(design_language)

Since UWP apps can run across a great variety of devices, you should design your UI responsive, so that it adapts well to different form factors and resolutions. Your base grid for best scaling should be 4x4 effective pixels.

Integrating into the Platform

UWP provides a large set of APIs that are available on each supported device family. If you want to use device family specific functionality, you can integrate its extension SDK and use adaptive code to call family specific functions. You typically use `Windows.Foundation.Metadata.ApiInformation` for testing if a specific family-specific API is present:

```
bool isHardwareButtonsApiPresent =
   ApiInformation.IsTypePresent("Windows.Phone.UI.Input.
   HardwareButtons");

if (isHardwareButtonsApiPresent)
{
   Windows.Phone.UI.Input.HardwareButtons.CameraPressed
      += HardwareButtons_CameraPressed;
}
```

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.

Services

Using the Microsoft Graph API[12] you can access a variety of Microsoft services at one endpoint.

Push notifications[13] are available that can also update the live tiles of your app. You can also consider using the freely available OneDrive cloud space[14] in your app.

Continuum & Cortana

Continuum allows the user to switch seamlessly between a mouse & keyboard and a touch-mode on desktop family devices. On mobile devices, Continuum even allows you to switch to a remote screen along with mouse and keyboard support, so that you can work "like a boss"[15] on your phone. The most important requirements for a great Continuum support is to design your app responsively and to test it with both mouse and touch input. For more details refer to *blogs.windows.com/buildingapps/2015/12/07/optimising-apps-for-continuum-for-phone*.

You can extent the "personal assistant" Cortana[16] by defining your own voice commands. You can show the results from the interaction either within your app or within Cortana itself.

12 graph.microsoft.io

13 msdn.microsoft.com/library/windows/apps/mt187203

14 dev.onedrive.com

15 microsoft.com/windows/continuum

16 msdn.microsoft.com/library/windows/apps/dn974233

Testing and Analytics

Test automation is integrated in Visual Studio. Use "unit test" projects for your modules and "coded UI tests" projects for your UI.

The Windows Store delivers various analytics statistics such as downloads, acquisitions, health, and more[17].

For developers wishing to collect runtime data and analytics, there are several options. Flurry[18] and Google Analytics[19] provide analytics tools and services that are compatible with Windows. Developers can also use the Microsoft Application Insights service[20]. There are robust performance monitoring tools available in Visual Studio for monitoring the performance during development.

Distribution and Monetisation

Distribute your apps through the Windows Store that supports more than 200 countries and regions.

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 Center[21]. 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 store account also allows you to

17 msdn.microsoft.com/library/windows/apps/mt148522

18 flurry.com/flurry-analytics.html

19 googleanalyticssdk.codeplex.com

20 azure.microsoft.com/services/application-insights

21 msdn.microsoft.com/library/windows/apps/dn764944

submit apps to the Azure and the Microsoft Office stores. The fee is waived for students in the DreamSpark[22] program. You can use the Windows Certification Kit that are both integrated with Visual Studio to test your application locally before you submit them.

Apps are managed by customer, not by device. So a user can install your app across a variety of devices or platform families, such as a desktop PC and a phone. If you create a Universal Windows Platform app you can choose if it only needs to be bought on one device family or if it needs to be bought again on each used device family.

Learn about monetisation and promotion options at *dev.windows.com/monetize* and *dev.windows.com/store-promotion*.

Learn More

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

The development team posts on their blog[23] and their Twitter account *@wpdev*. For a large collection of independent developer and designer resources, visit *reddit.com/r/wpdev* and *windowscentral.com/developers*.

You can extend your components, behaviours and other tools with various commercial and open source toolkits. Popular examples include Telerik[24], Coding4Fun[25], Cimbalino[26], MVVM

22 www.dreamspark.com

23 blogs.windows.com/buildingapps

24 telerik.com

25 github.com/Coding4FunProjects

26 cimbalino.org

Light[27] and MvvmCross[28]. For inspecting the visual tree, bindings and properties of XAML-based user interfaces at runtime, xaml spy[29] is available.

Microsoft also provides a series of online videos for beginners and experts for all kind of topics in their Virtual Academy[30] and on the Channel19 website[31] - including recordings from all Microsoft developer conferences.

Find sample code on *dev.windows.com/samples* and in various Codeplex[32] projects. The How-to guides for app developers[33] provide a good overview about planning, designing and developing Windows apps.

If you are porting an existing app from iOS or Android you can find help at *msdn.microsoft.com/library/windows/apps/mt238321*.

27 mvvmlight.codeplex.com

28 github.com/MvvmCross/MvvmCross

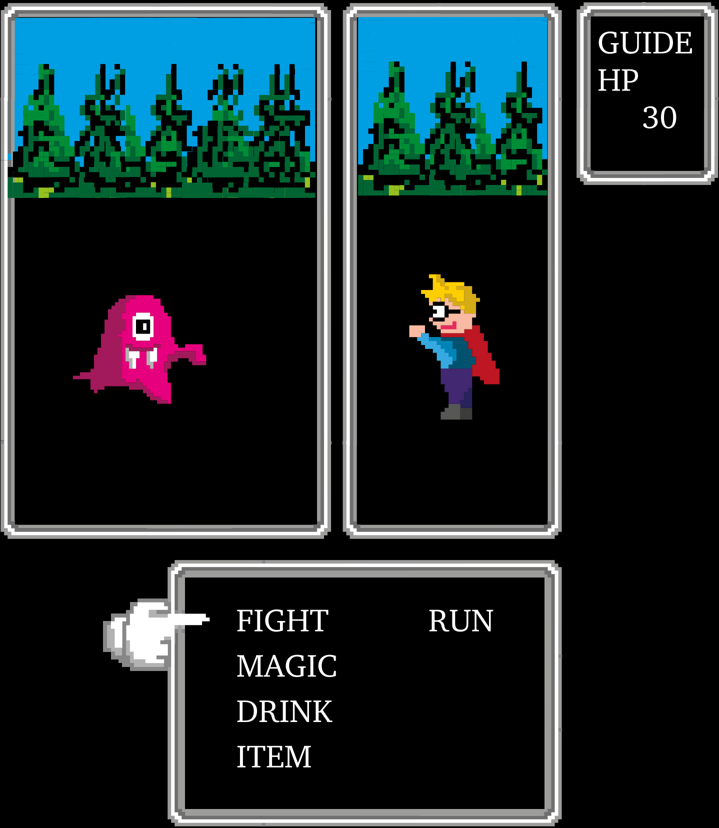
29 xamlspy.com

30 microsoftvirtualacademy.com

31 channel9.msdn.com

32 codeplex.com

33 msdn.microsoft.com/library/windows/apps/xaml/mt244352


GUIDE
HP
30
FIGHT
RUN
MAGIC
DRINK
ITEM

BY Robert Virkus

Going Cross-Platform

So many platforms, so little time: so why not go cross-platform? There are more than enough platforms to choose from: Android, iOS, Sailfish OS, Tizen, Ubuntu and Windows are waiting for your apps.

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	Supported Natively [1]	Supported Optionally [2]
C, C++	Sailfish OS, Ubuntu, Windows	Android (partially, using the NDK), iOS (partially)
C# & Visual Basic	Windows	none
Java	Android	none
JavaScript	Tizen, Ubuntu, Windows	none
Objective-C	iOS	none
Swift	iOS	none

1 for example either the primary or only language for creating applications

2 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 as well.
— **Interpretation/Virtual Machine**: 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-independent.

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.

Nowadays most platforms use a variation of the flat design as pioneered by Windows Phone in 2007. However there are many differences and subtleties between the platforms, so porting over the exact same look and feel would result in an awkward looking app on other platforms.

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' and go native.

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 customising 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 / home screens varies a lot between the platforms; on iOS you can only add a badge with a number to your app’s icon, on Windows 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 realised differently among operating systems. On Android, Ubuntu 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 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

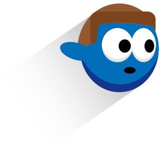
Push services are a great way to give the appearance that your application is alive even when it is not running. In a chat application you can, for example, send incoming chat messages to the user using a push mechanism. The way push services work and the protocols they use, again, are realised differently and use different data sizes on each platform.

In App Purchase
Today's most important monetisation option is in-app purchases. Needless to say that this works differently across platforms. See the monetisation and the platform-specific chapters 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, can offer better revenues. Again, these vendor services work differently between the platforms. The monetisation 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.



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, there are many commercial and open source components available for all platforms.

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 Xamarin[1] and Lua[2]. 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. Sometimes player concepts use a "least common denominator" approach to the offered features, to maintain commonality among implementations for various platforms.

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 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.

1 xamarin.com

2 lua.org

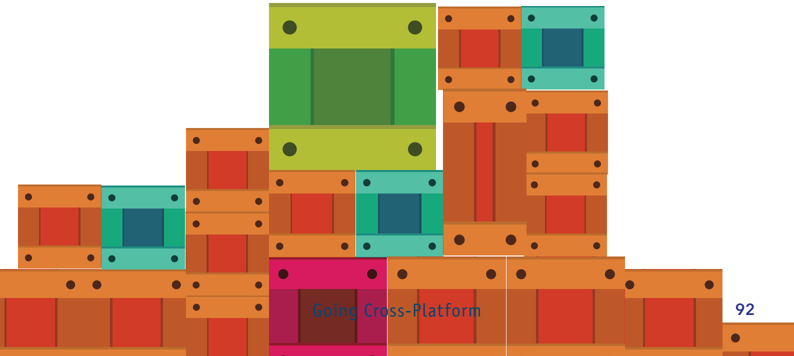
(Hybrid) Web Apps

Hybrid web development means to embed a webview within a native app. The standard for hybrid apps is the open source tool Apache Cordova[3] (formerly known as PhoneGap). 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. Read the web chapter to learn more about mobile web development.

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, iOS and Windows. 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.

3 cordova.apache.org



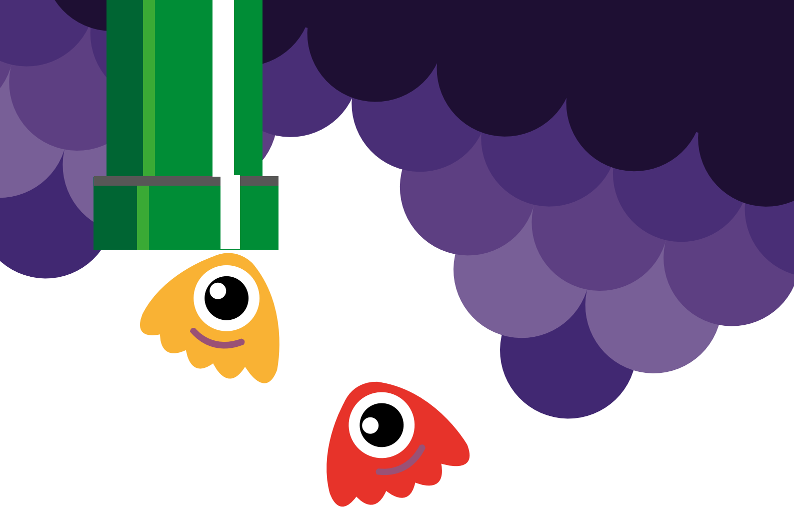
Finding the Right Cross-Platform Framework

There are so many cross-platform solutions available, that we decided not to list them in this book anymore. If you are still interested, please check previous editions of this guide. We hope to resurrect our list in a digital form in the future. For a benchmark of the available frameworks refer to the research-2guidance[4] report.

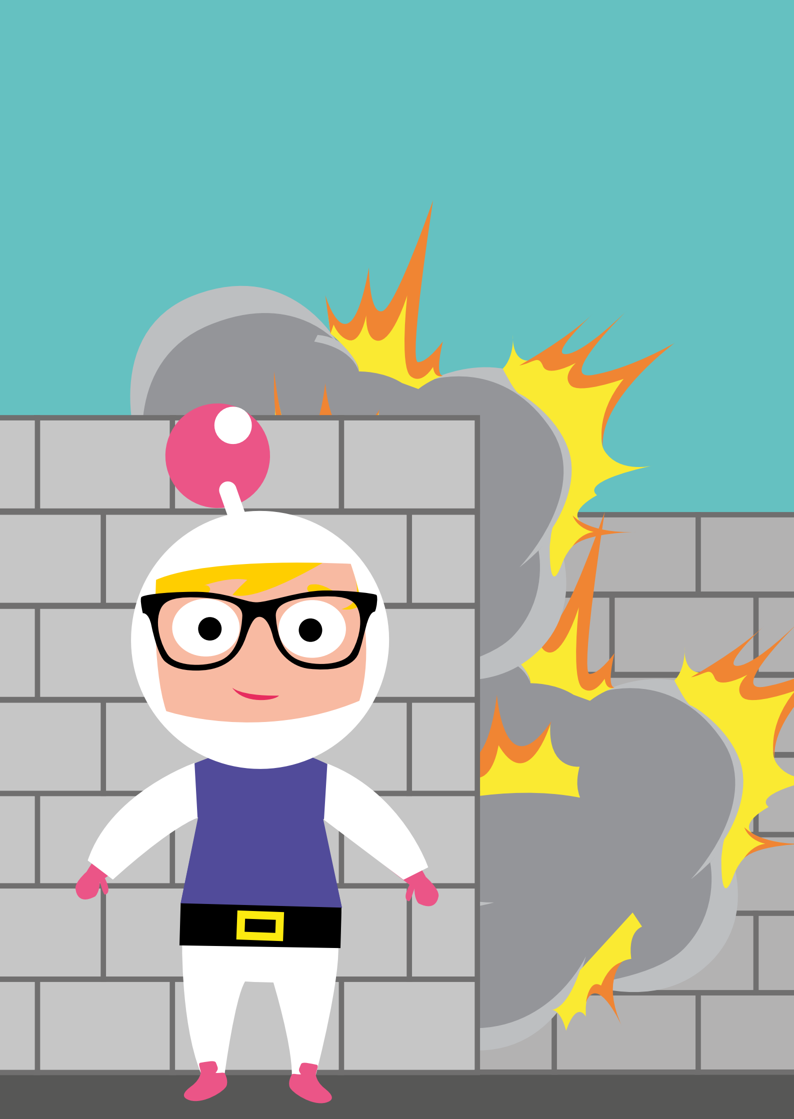
Here are some questions that you should ask when evaluating cross platform tools. Not all of them might be relevant to you, so weight the options appropriately. First have a detailed look at your application idea, the content, your target audience and target platforms. You should also take the competition 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?

4 available at *research2guidance.com/cross-platform-tool-benchmarking-2014*



- 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?
- What's the performance of the solution?



BY Daniel Kranz

Mobile Sites & Web Technologies

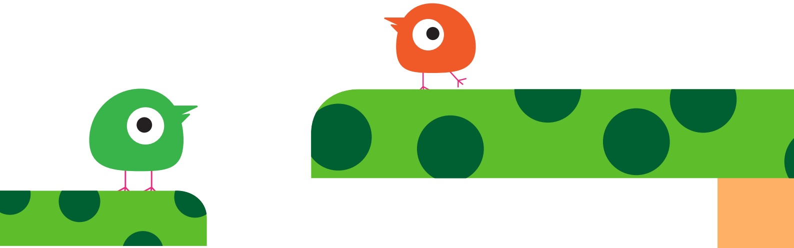
The continuous development of web technology coupled with an increase in Internet-capable devices promises a great future for those catering to the ever-increasing mobile web audience. Global mobile Internet traffic is growing rapidly and has already surpassed 52% in 2015[1]. The share of time spent browsing the Internet by device (mobile, tablet, desktop and Smart TV) varies greatly across the globe. Most regions where mobile Internet traffic has already surpassed desktop Internet traffic are developing and emerging markets. Developed markets such as the US with a high smartphone penetration follow suit. Worth pointing out that whilst feature phone markets see considerable time spent within mobile search and mobile site browsing, smartphone penetration heavy markets see the highest engagement within apps. Since May 2015, mobile apps account for more than half of all digital time spent in the US[2].

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. 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. Additionally, they are arguably easier to learn than some of the rather complex languages needed for

1 statista.com/topics/779/mobile-internet

2 comscore.com

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 does not only save 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.

Monetisation of mobile sites can prove tricky as well, since users expect to access mobile sites free of charge. The most common monetisation 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 monetisation tools by contrast offer easy set-up and a high level of security for the end-user.

If monetisation 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 learn more on how to create this type of so-called "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 SMBs[3]:

- **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

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

- **Make it local** - including functionality that helps people find and get to you
- **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 are built with input from your audience.

Google has also rolled out changes to its mobile search results and has announced that it will penalise sites that are not in line with its recommendations. Have a look at Google's developer site[4] for more up-to-date information on how to optimise your mobile site.

HTML5

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 optimisation based on screen size make HTML5 an appealing alternative to native app development.

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

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

Ex-Facebook CTO Brent Taylor once described 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.' Unfortunately this is still true, although the situation is continuously improving.

For more info on browser compatibility, check out the HTML5Test online[5]. The site provides both an overview and in-depth analysis of which HTML5 features are supported by which browser. Facebook has also developed ringmark[6] 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 the last couple of years saw rapid adoption of HTML5. ABI research estimates that mobile devices with HTML5-compatible browsers will total 2.1 billion worldwide by the end of 2016.[7]. Operating systems will gradually increase support for HTML5 features and browsers to increase overall adoption and speed. Open-source platforms such as Sailfish, Tizen and Ubuntu should also help to speed up adoption.

5 html5test.com/results/mobile.html

6 rng.io

7 www.abiresearch.com/press/21-billion-html5-browsers-on-mobile-devices-by-201

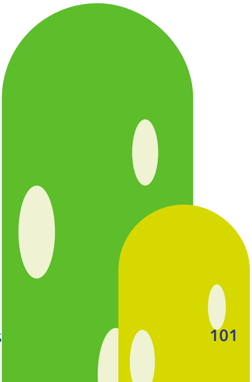
Fragmentation Needs Adaptation

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 optimise 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 optimising 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.



Client Side Adaptation

Responsive Web Design

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.

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 optimisation (e.g. Top 5)

Coupled with server side components and therefore generally referred to as RESS (Responsive Design + Server Side Components), responsive design can also be used to deliver more complex desktop and mobile sites.

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 Mobile[8].

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

8 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 WURFL[9] and DeviceAtlas[10]. The main drawback of device databases is that the majority is only available as part of a commercial licence.

Pro

- Most commonly used solution (Google, Facebook, Amazon and alike)
- Maximum control
- Device optimisation 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)

Responsive Web Design + Server Side Components (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

9 wurfl.sourceforge.net

10 deviceatlas.com

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 organisations.

RESS/ Hybrid Adaptation solutions are available commercially from companies such as Sevenval[11] and Netbiscuits[12], and as community-backed cloud solutions, for example FITML[13].

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 cannot influence, the following sections provide tips to reduce transfer size, content and HTTP requests to minimise load time and improve performance.

Reduce Transfer Size

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

11 sevenval.com

12 netbiscuits.com

13 fitml.com

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 is 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 minimised 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 slideshare[14].

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

Hybrid and Cross Platform Apps

Armed with HTML5, CSS and JavaScript skills and keen to give app development a go? Hybrid app development could be the route for you. Adobe's Phone Gap[15] is just one example of cross platform frameworks and tools available to build, distribute and update across a number of sites and apps conveniently controlled via one single system. Please also read the "Cross-Platform" chapter in this guide to learn more about your options in that area.

Pro

- Use your existing skills
- Single code base for multiple platforms (iOS, Android, Windows,...)
- Cost and time savings when developing for multiple platforms
- Distribute both via app stores and mobile browsers for additional reach

Con

- Expect performance issues when emulating complex native functionality
- Mimicking native user interfaces will add additional time and effort to your project
- Risk of being rejected by app stores (mainly Apple) if your app does not feel native enough
- Some device and operating system features might not be supported

15 phonegap.com

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 online[16]. The set of devices can be refined by analysing 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, WebDriver[17] 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 behavioural 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 behaviour 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.

16 gs.statcounter.com

17 seleniumhq.org/projects/webdriver

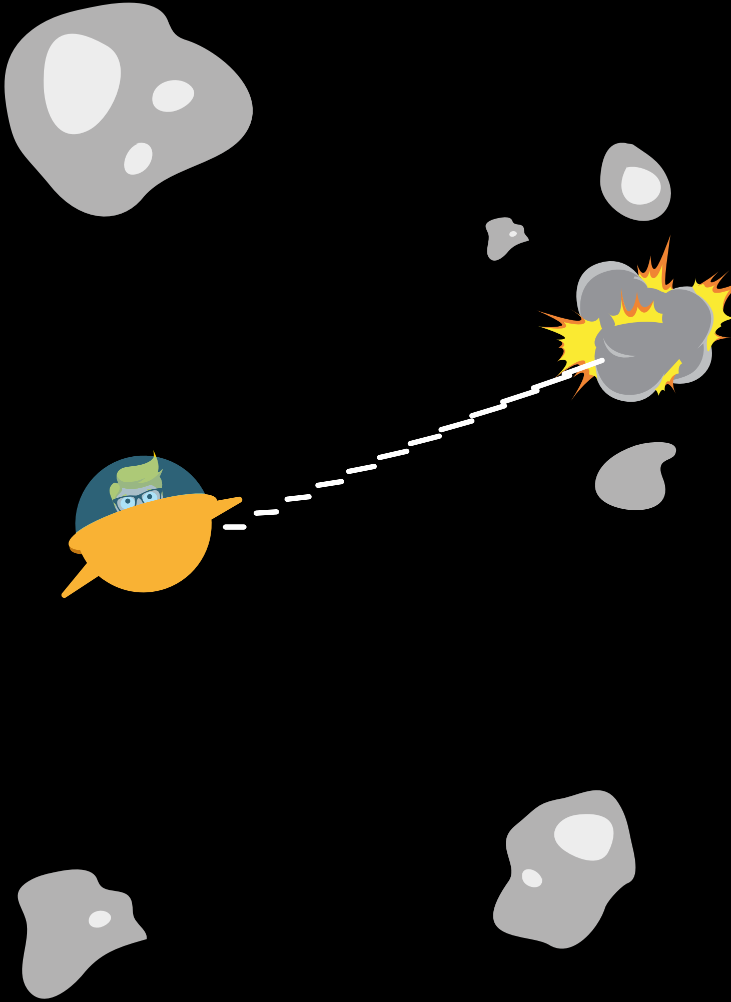
Learn More

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 organisation that defines web standards): *w3.org*

Books

- **Mobile First** by Luke Wroblewski
- **Adaptive Web Design: Crafting Rich Experiences with Progessive 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



BY Ian Thain & Davoc Bradley

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 & big 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 report of Q3 2014[1].

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
- Analysing and utilising Big Data

Enterprise Strategy

Many companies nowadays have a Chief Mobile Officer (CMoO) or have extended their CIO position. It is their job to co-ordinate 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

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

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 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 increase the adoption and usage of the apps created.

In commercial adoption terms enterprise app development is mostly mainstream now. The question a company writing third party enterprise apps, or a development manager keen to adopt an internal mobile enterprise strategy used to be “This all sounds great, but why do we need it?”. This has now become "Mobile will give us a competitive advantage and empower our workforce" which is a compelling reason 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, on various devices and know how they will aid, shape and empower your enterprise.
- Create an ADS (Application Definition Statement) for each app, specifying purpose and intended audience.
- Create a budget for devices & device upgrades.
- Create a plan for an application & device management strategy & security infrastructure.
- Create a plan for an app dev team using a future proof development architecture - such as a MADP, Frameworks etc.

Mobilising Existing Systems

If you are already providing a system to customers which has not yet been mobilised, 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 implementing the solution. Common reasons to mobilise 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 mobilise 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 makes 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 mobilised.
- 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 multiple 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**: *go.sap.com/product/technology-platform/afaria-mobile-device-management*

Mobile Application Development Platforms (MADP)

Usually, one key element of enterprise applications is data synchronisation. The mobile devices have to be refreshed with relevant or up to date data from the company's servers and the updated or collected data has to be sent back. The scope of data access is determined by the job responsibilities of the user as well as by confidentiality policy. In any case synchronization has to be secure, as corporate data is one of your most prized assets. Furthermore, a company-wide accepted app will be multi-platform.

To compensate the shortcomings of the native SDKs as well as the common multi-platform solutions in these regards, you might want to consider evaluating Mobile Application Development Platform (MADP) solutions. MADPs are mobile development environments that provide the middleware and tools for developing, testing, deploying and managing enterprise apps running on multiple mobile platforms with various existing back-end datasources. Their aim is to simplify development and reduce development costs, where skills must be maintained for multiple platforms, tools and complexities, such as authentication and data synchronisation.

Available solutions include:

- **IBM MobileFirst Platform**: *www.ibm.com/mobilefirst*
- **Kony KonyOne**: *www.kony.com/products*
- **Pega Amp**: *www.pega.com*
- **SAP Mobile Platform**: *www.sap.com/smp*
- **Spring Mobile Solutions**: *www.springmobilesolutions.com*

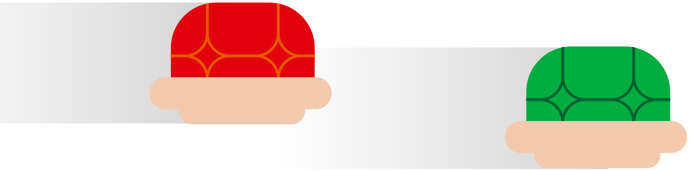
Security In Enterprise Apps

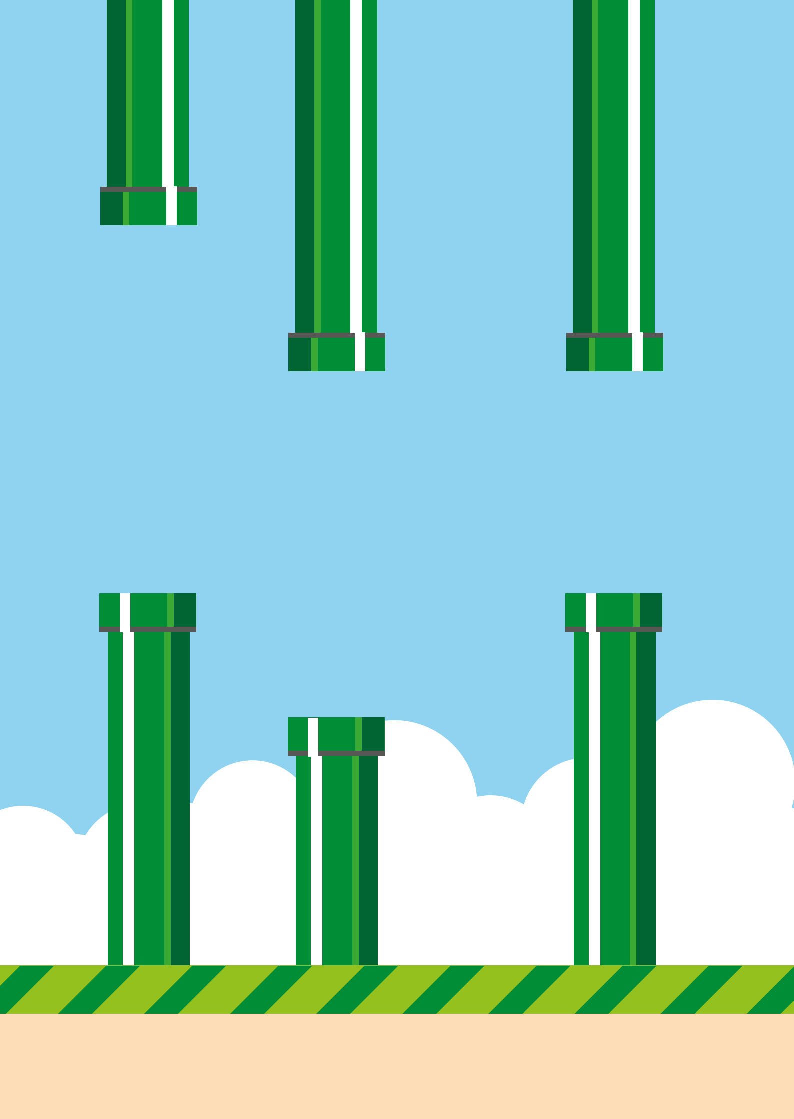
One of the main functions of any IT department is to ensure that all aspects of the company infrastructure is secured against attack so that there are no data leaks and no data is compromised or stolen. As mobile devices are an extension of a company's IT infrastructure, all enterprise apps must be designed to ensure that they cannot be used to illegally gain access to a company's internal network. As an enterprise app writer you will usually be asked to conform to standards which a company has laid out in their security policies, so be prepared to answer questions about securing your app, such as data encryption, network communication and dealing with jail broken or rooted devices.

Many EMM providers actually enhance app security, using techniques such as app wrapping or providing an SDK which app writers can use. These features, and regular updates of these platforms, allow an enterprise to lock down their apps remotely and also keep up with the changing security landscape without needing to invest as much time and effort into security.

Key points for securing Enterprise Apps

- If using an EMM provider ensure they have the security required security features to meet your enterprise standards.
- When storing any data on the device ensure it is encrypted.
- When communicating with web services, always use https.
- In addition to using https, when communicating with web services ensure you perform end point checking in both the app and the web service to confirm that the server/device you are connecting with is valid.
- Always check that any settings your app is packaged with have a checksum to ensure that the values cannot be changed once shipped to the device.
- Do not allow the app to run on jail broken or rooted devices.
- Have a method for disabling the app if the app detects that it has been compromised.
- Ensure that all use of encryption complies to export regulations and any laws relevant to the region/s the app is being used in.





BY Oscar Clark

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 2017[1]. According to VentureBeat[2] 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 Android[3].

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 realise 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 realise 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.

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"[4]. He claimed that games consist of a world, activities and characters. We can change any of these for new

[4] mrbossdesign.blogspot.co.uk/2008/09/triangle-of-weirdness.html

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'[5].

We have to appreciate that what makes 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 standout 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 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

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

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. Let us 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 realisation 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 (FPS) 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 recognise 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 Unity[6], the reported spend by paying players who spend less 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 realisation 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 realisation that it is not only our hero character who is going on a journey, but our player as well. This journey consists of several stages:

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

1. 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.

2. Engagement & Onboarding
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 recognisable 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.

3. Scaffolding

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”.

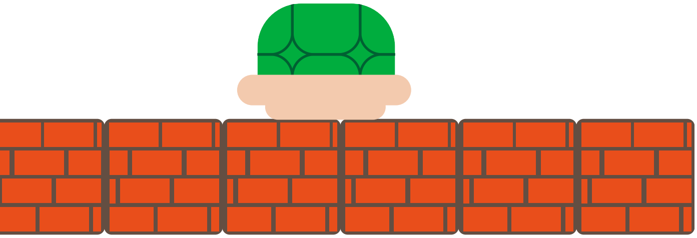
4. Endgame

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 by also in terms or 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 behaviour. In practice the majority of True Fans are rational people who have made your game their principle hobby. Addictive Behaviour is always detrimental to the individual and we should do everything we can to help anyone with these kinds of issues.

5. Churning

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?



Analytics and Game Flow

Designing your game from the perspective of the player's lifestages is really important, but in order to leverage the benefits of that strategy we need to know about how the player responds to that experience and that means we need to capture player activity in our game. For example, we might wish to capture that a player was shot in a FPS game. We might give that a name, say 'PlayerShot' and then store a range of variables with that event. This could include the date and time that the hit was resolved, an anonymised 'PlayerID', the 'X,Y,Z' coordinates of their avatar, the damage taken, and the anonymised 'PlayerID' of who fired the shot as well as the sessionID for that game.

Importantly we do not have to capture everything. There are some kinds of data which are static, reference information. For example, the specific position in a specific map. As long as the version of the map used at that time is known then X,Y,Z coordinates alone can be used to create a heatmap later. We can also infer a lot of data from other events as long as there is some connecting information. For example, we do not need to capture the level that the player is using for that game in every event or even a list of all the players in that session. We can capture that information with a specific 'Start Session' events and use the associated Session ID to allow us to identify everything that happened in that specific game session.

But even if we would want to capture everything, we will never be able to do so: The data we collect will always be incomplete, for example if the battery dies or the player switches to a phone call – we will probably not get the last upload. This is less of a problem with a server-based game but it is never 100% and the compromise is that the game cannot be played offline; impacting our chance for them to create habits of play.

Notice we want our players to remain anonymous. We do not want or need to spy on our players but we do need to understand how the game plays across all players.

What Events Should We Capture?

When considering which events to track, think of them in terms of the timeline in which the player might encounter them. There will of course be a 'First Time Player' experience, but it can also be useful to map out more commonly experience 'Engaged player' session flow. We are not necessarily mapping every button press directly. Instead you should look for moments where there are meaningful choices. There is an approach used by the food industry called HACCP[7].

7 www.develop-online.net/opinions/navigating-the-hazards-of-game-data/0187815 In essence the point is that we are looking for the 'Hazards' such as whether they churn (i.e. leave the game) but also trigger points for more positive action such as paying for an IAP or watching a video Ad.

Some typical events worth tracking might include the following:

- **GameMenuLaunch:** AnonPlayerID; TimeIconLaunched
- **SessionLaunch:** TimeSessionLaunched; AnonPlayerID(s); SessionID; LevelIDSelected; OptionSelected
- **SessionStart:** TimeSessionStarted; AnonPlayerID; SessionID;
- **ObjectiveSet** TimeObjectiveSet; AnonPlayerID; SessionID; ObjectiveID;
- **ObjectiveMet:** TimeObjectiveMet; AnonPlayerID; SessionID; ObjectiveID; Score; Reward; XYZLocation
- **TargetHit:** TimeTargetHit; AttackerID(AnonPlayerID?); SessionID; TargetID(AnonPlayerID?); Damage, XYZLocation
- **PlayerDeath:** TimePlayerDeath; AnonPlayerID; SessionID; XYZLocation
- **LevelComplete:** AnonPlayerID; SessionID; ObjectiveID; Score; Reward; XYZLocation

From creating events in this way we can infer a huge amount of information. For example, if we want to know the percentage of players who complete a level we can count the number of 'GameMenuLaunch' events with the number of 'LevelComplete'. But we can also get smarter with our analysis. We can look at how many people completed a specific ObjectiveID in a specific 'LevelIDSelected' and compare that to the number of 'LevelComplete' in the subsequent level to find out if skipping objectives in earlier levels have a particular impact on performance later.

Data capture like this can be very powerful, however the way you capture information will be inherently biased towards your understanding of the way the game is 'meant' to be played. This means that we have to constantly review our metrics. The biggest issue is that we cannot capture what players 'will do' or 'might have done'. That might seem obvious but we will be using insights from analytics to inform our design decisions. If it is true that only 2% of players spend money in a Free2Play game then that means we do not know what would have triggered the other 98% to spend. There might not be anything that would have convinced them but the point I am making is that the data we capture cannot ever be complete and that means we have to consider statistical significance and be very wary of assuming causation rather than correlation. Looking at the life cycle of a player helps us mitigate this because we can breakdown each decision stage and look for ways to increase the likelihood of converting players at each stage. It is all about asking the right questions at the right time.

Free vs. Paid

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.

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 Free2Play 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. We have to find a way to engage the player with the content we are creating to delight them.

Done well, a Free2Play 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, Free2Play 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.

Free2Play 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 behaviour 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 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 behaviour. Games are not gambling, this activity 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 recognised problem, it is something that comes under the term of behavioural 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).

In-App Purchase

The easiest way to understand Free2Play is to realise that these games have simply taken the retail side inside the game. That means that the people who know the game and its players best (i.e. the developers) can identify items that players will love that compliment and enhance the experience for all players; and sell them directly to the audience that loves that game. However, many of the current games have focused on the relatively simplistic implementation of some kind of energy or similar 'blocking' system where you are restricted from playing the game until either time has passed or you spend money.

This can be highly unsatisfying, depending on the game but fortunately it is not the only way to build in game monetisation and arguably has lead to unhelpful discussions about the ethics of Free2Play design. However a South-Korean study[8] revealed an important truth in games purchase behaviour: Players do not spend money because they are happy or (sadly) because they like the developers. They spend because they expect future value.

As a result we need to know what players need at their different life stages and satisfy that. Just like any purchase decision, in-app games offerings have to overcome the potential objections ('Customer Behaviour as Risk'). The following examples show how some games attempt to do this.

— **Unfinished Business:** Games like Kim Kardashian Hollywood do an amazing job with the narrative progression and the format of what are essentially ‘Cookie Clicker’ tasks to create an entertaining sense of unfinished business. The gameplay may appear to be limited but player engagement

8 staffweb.hkbu.edu.hk/vwschow/lectures/ism3620/c26.pdf

is in fact very real – always leaving the player wanting more.

- **Continued Relevance:** Games like the VEGA Conflict show items the players have already unlocked and what they will be able to unlock later in the game. There is a comparison of stats and costs to demonstrate the continued relevance of what we have already unlocked as well as why we should continue playing to unlock more.
- **Social Capital:** Too many games ignore the impact of social visibility on not only purchase behaviour but satisfaction levels and virality. Especially where players have the ability to personalize their experience. This was a key to revenue for Playstation®Home (now shutdown) with examples like the 'Gold Suit' offering its wearers social kudos for owning them.

Many games will have a high proportion of spending happening in the first few hours of player starting a game, however this is wasted if those players do not spend more than once. If that happens for most of our paying players, then we should consider our monetisation design a failure. As designers we always have to consider why one player making a purchase makes the experience better for everyone and why they would do that again. And again. And again...

Developers looking to provide a greater level of utility (or expected value) from the goods they sell in the game are starting to take a more game design approach to the kinds of goods we offer players. I like to break these down into four categories:

1. **Sustenance:** Goods we require to continue playing
2. **Shortcuts:** Goods which speed up the actions we are performing
3. **Socialisation:** Goods which are primarily about social capital
4. **Strategy:** Goods which open new playing options

These goods can come in various forms:

- Consumable – a one-time use item
- Capacity – something which enhances growth/play
- Permanent – a permanent upgrade or unlock item
- Generators – an increase in the supply of a consumable

Looking at the player's journey of your game, you should then be able to identify a point in the game mechanic or the context loops of play (perhaps even the metagame) where you can combine these types and forms. For example in a driving game fuel is a Consumable/Sustinence good; but you can also consider selling an increase in the size of the fuel tank (a Capacity/Sustinence good) or to offer a petrol pump which delivers X Fuel per day (a Generator). Then when a player upgrades to an aspirational car part of the strategy of play can be influenced by whether this uses more or less fuel that the other cars you own.

Critical to your success for in-app purchase comes not just from making what you sell desirable but also making sure that it can scale. We want our players to buy again and again and again and that only happens when we deliver predicable consistent and still suprising value. If our game has only one measure of success then how can we create scalable value. Take a first person shooter, if I am going to spend money at all then why would I buy anything other than the biggest most powerful gun I can afford? and when I do that any previous purchase of a gun is redundant. But! if guns have contrasting benefits like armour piercing, area affect, silencer, etc then we create reasons for buying a wider armoury of weapons.

For more detail on this check out my blog post on the Unity website[9]

Advertising

Advertising in games is a highly successful way to complement the revenues from in-app purchases. Indeed for some games they can provide the principle revenue model. Traditional ads like banners or interstitials work by providing an overlay of some kind which interrupts the full playing experience. However some of the biggest new games in the last year have been using opt-in incentivised video ads. In this model the player chooses to watch the game in order to get a small benefit in the game.

There are a couple of factors which affect players' willingness to actively choose ads. The ad content itself must be relevant, i.e. showing a game player an ad for another game, and they should also be short – somewhere between 15-30 seconds. However, there usually has to be something in it for

9 blogs.unity3d.com/2015/06/23/design-driven-in-app-purchases-creating-sustainable-monetisation

the player. That might be a number of things from coins or resources in the game, to extra lives. Yet these rewards also have to be relevant to the player and repeatable – there must always be a reason for the player to watch another ad. If I unlock a major item or a new level by watching one video, that is a great value but unlikely to create the scale of views needed to drive enough revenue to make that worthwhile. Equally, if watching videos unlocks enough power-ups to imbalance the game then you have broken the reason to play.

Angry Birds Go uses video ads to allow players to obtain a free boost, but only at the start of the race. This does not overpower the experience but it does require a player to switch their attention from the game strategy to the video playback. The motivation is clear: If I want a boost in my next race then I will get it by exchanging my time watching this advertisement.

Sonic Dash offered an option with the game to continue their existing run after 'dying' either using a 'revive' token or by watching a video ad. Interestingly the timing of the ad (around 15-20 seconds) also allowed the player to take a short break from the intensity before restarting play, which can help performance as well as motivation.

Yet another approach can be seen in Crossy Road where they offered players a small amount of extra coins for watching the video. These coins were used in exchange for a random character (all of which add delight but without changing the gameplay). Interestingly the random nature of the benefit meant this did not cannibalise the purchase of those characters - in fact it made the decision to purchase easier! Hipster Whale did something else that was interesting. They capped how often you could see an ad to a predictable time frame, making it even more desirable.

What is important is that the payoff for watching the video is intrinsically part of the game itself. There is no separate incentive to download the advertised game, but if they choose to then it is because they liked the look of the game. That is what at the end of the day provides the revenue and at the same time reinforces their engagement with your game.

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 monetisation 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 Pitch[10] or the Indie Awards at Casual Connect[11]. If you can get the attention of YouTubers that may also help.

Spending money on advertising can help, but it is important to realise 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 aware-

10 www.pocketgamer.biz/events

11 indieprize.org

ness 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.

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 Ads[12], *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.

2015 marked a point where the role of social engagement with games hit critical levels and YouTube personalities like Pewdiepie and Yogscast are having significant impact on the

12 unityads.unity3d.com

take up of games, including mobile. Teams like Hipster Whale and Seriously have made engagement with these important celebrity influencers to help propel their games to the top of the app store. However, like any medium this is becoming increasingly commercial but it remains important to consider the audience and why a YouTuber engaging with your game will be entertaining for potential players.

On the eSports side it has been interesting to see that as well as the huge audiences on online channels such as Twitch that even television channels like ESPN have started to take mass audiences watching games as seriously as other more physical sports. The level of talent and professionalism in the eSports market is now significant and game developers are starting to consider how this will impact game design. However it is still the case that there are very few mobile games which can legitimately claim to have gained a strong enough following.

In the end despite all the differences in the details, mobile is like any other platform. We have to acquire, retain and monetise 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.

BY Alex Jonsson

Mobile Development & the Internet of Things

A decade ago there were two giants; the mobile industry and the Internet of Things (IoT)[1] lived separate lives without speaking that much to one another. Each of them enjoying trillion dollar turnovers, yet with little shared knowledge and necessary insights on how to ultimately provide end-to-end services for customers, as that would have required both giants' concurrent attention. If the major players industrial internet had early on followed the route paved by the consumer market, and relied more on mobile terminals like mobile phones and tablets to deliver services to end-customers, this chapter would have brandished an abundance of tools, platforms and service available today. However, this has not been the case and only in the very last few years has there been any greater interest in IoT involving modern smartphones, resulting in a lesser albeit interesting solutions worth disseminating in these pages.

One of the drivers we see, is that several traditionally closed hardware systems are opening up their service APIs, and move codebases to GitHub[2] and other open code repositories in order to simplify developer access to industrial systems and services. It looks very promising having big industrial corporations like Ford, GE, Bosch, Cisco and Siemens migrating towards more open standards, and exposing their bespoke product and service APIs to developers accustomed to IP-based technologies. For us in the mobile industry, we see an increasing interest for hooking up phones to connected things is becoming more

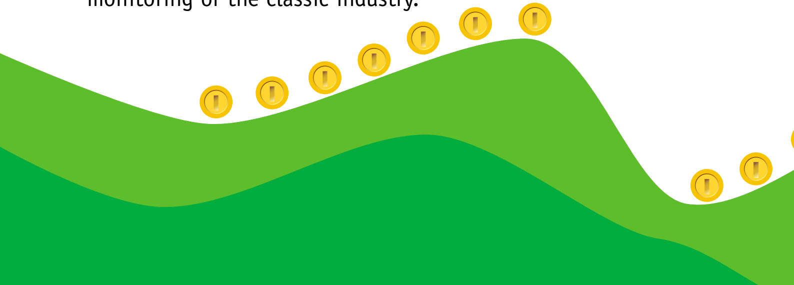
1 also known as M2M (machine-to-machine), embedded or industrial internet

2 github.com/

commonplace, and many hardware devices are found to rely on supporting mobile apps to obtain their full functionality range. Moving out of the self-contained model allows them to release products earlier, and ability to add a feature over time by upgrading firmware, in tandem with the functionality and number of mobile apps. The concept of releasing in this green banana fashion has proven very useful also from a competitive point-of-view and has spread from digital into the physical world, thanks to dynamic software architecture, over-the-air updating not to mention 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 localisation 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, visualisation, storage, off-line functionality and well as means of authentication are other probable roles. 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 shows that there is a considerate interest in IoT apps. At the time of this writing, the worldwide mobile industry has grown 13% the last year alone and still is pretty much a two-horse race between Android and iOS. The industrial side will most probably mimic this relation even though there will be more Apple devices represented in the US and European markets. Many IoT developers continue to use Apple's and Google's toolkits for development, while several newcomers have been made available for developing mobile apps by other means, and using other languages. Here are some examples of typical tools for development within IoT:

- **IBM Mobile First**, Enterprise and industrial tools, formally known as Worklight, more focus on iOS: *ibm.com/mobilefirst*
- **Ionic**, front-end development tools, addressing multi-platform UI: *ionicframework.com*
- **Xamarin**, delevoment in C#, support many native features: *xamarin.com*
- **Evothings**, rapid development suite for native/hybrid apps using JavaScript: *evothings.com/download*
- **Waygum**, an application platform for mobile IoT, with mobile front-ends: *waygum.com*

The Devilish Details

Every market segment of IoT; from wearables to real estate automation, from medical applications to surveillance and fleet management has their own special needs and challenges; offline usage, large datasets, need for strong encryption, real-time interaction without delay or high demands on bandwidth and/or accessibility on a global scale. No single tool or library covers them all. The context in combination with the wrong feature set may limit your implementation options. Contextually adaptive functionality, often in tandem with open standards is the way to prevail in the early phases of this market, until it has hit the level of maturity that many other segments in the mobile ecosystem have reached.

"What makes an IoT app?", someone might ask at the dinner party. "Isn't it just any ol' database client on a smartphone, so what’s the big deal?" Well, cloud access is an important component, but not the only important one. The devil lies, as usual, in the details. Let us say you are auditing your power consumption in your house, and there is a gadget connected to your fuse box. It speaks continuously to a data collecting service via a RESTful interface over HTTP, and you can as an effect access a web page riddled with graphs via your tablet. But now 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 mobile web project's budget. Then a native solution using Apple's and Google's tools, or for many cases a hybrid application 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 need to make an app with several versions (function sets, languages, various UIs et cetera). Web and scripting technologies are intrinsically easier to grok for the novice developer[3]. They also enable faster development also for the journeyman developer to create a decent UX, and then select from a variety of industry-strength third party libraries to implement the intrinsically native parts of your app.

One of the more popular base technologies for hybrids is the Apache Cordova project, sibling to its commercial incarnation Phonegap[4]. Several commercial hybrid SDKs use Cordova as one cornerstone, noteworthy Mobilefirst (IBM), Salesforce One (by Salesforce), Evothings Studio by Evothings, Appgyver 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.

3 Check the cross-platform and the web chapter for more information about the general potentials and limitations of the hybrid approach

4 phonegap.com

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 XMPP[5], MQTT[6] and CoAP[7]. Smartphone apps 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 client[8]. MQTT can run both over raw TCP/TLS sockets and Websockets, which allows also this format to run inside a web browser.

To be able to interact over low-level TCP and UDP based networking, transport security et cetera, technologies like Chromium sockets (i.e. Berkeley sockets nicely wrapped for javascripters) available as plug-in technology for Apache Cordova needed to be introduced.

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. Coming from the old embedded world can be scary for an organisation who has been enjoying bespoke networking not sharing cables with anyone, and then in the flash of an instance after connecting a HTTP gateway to the old M2M system, get all the possibilities, horrors and problems that the internet residents have seen

5 xmpp.org/

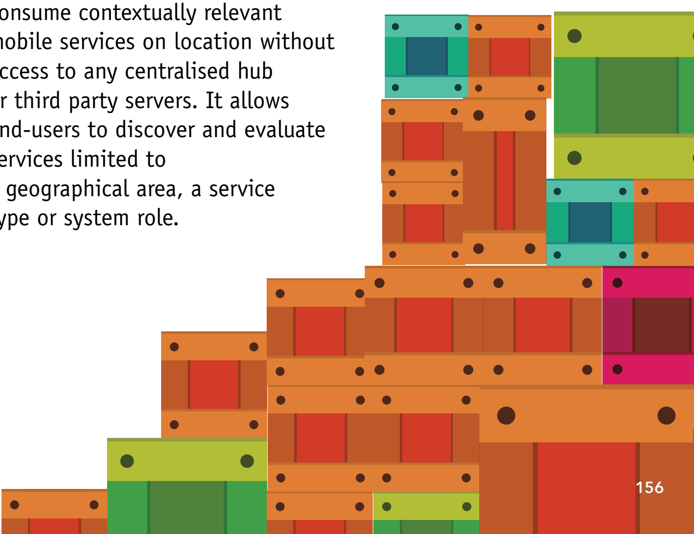
6 mqtt.org/

7 tools.ietf.org/html/rfc7252

8 eclipse.org/paho/clients/android/

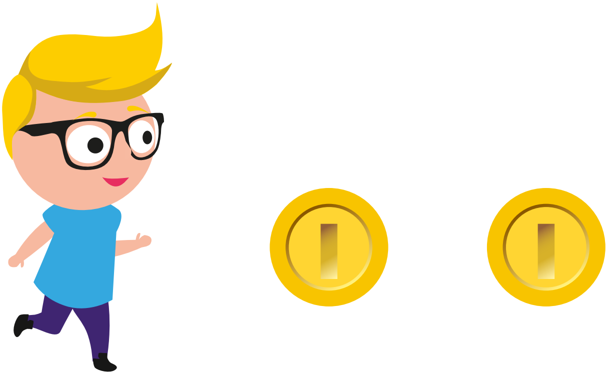
for more than 20 years. Securing the messages, securing the channel is the way to obtain a peace of mind.

So in the wake of learning users more than a this database client, “ löschen und ersetzen durch: “IoT apps proved over time to be more than simple web-based database clients, and will be doing much more than visualising server-side generated data views. A second wave of apps is coming our way in where IoT mobile services o phones converse directly over short-range radio, using low-level IP-based protocols for sensor data and telemetry messaging with a minimum of overhead. The prevailing standard here is Bluetooth Smart, which lately has acquired an increased sense of security as well as meshing capabilities. Two of the most interesting application thereof are both chipsets allowing the bluetooth radio to be in announcement (broadcasting) mode and connect services concurrently, which takes Bluetooth as a concept beyond the somewhat limiting one-phone-one-device concept. The other being the open Eddystone bluetooth beacon format, assisting in having users consume contextually relevant mobile services on location without access to any centralised hub or third party servers. It allows end-users to discover and evaluate services limited to a geographical area, a service type or system role.



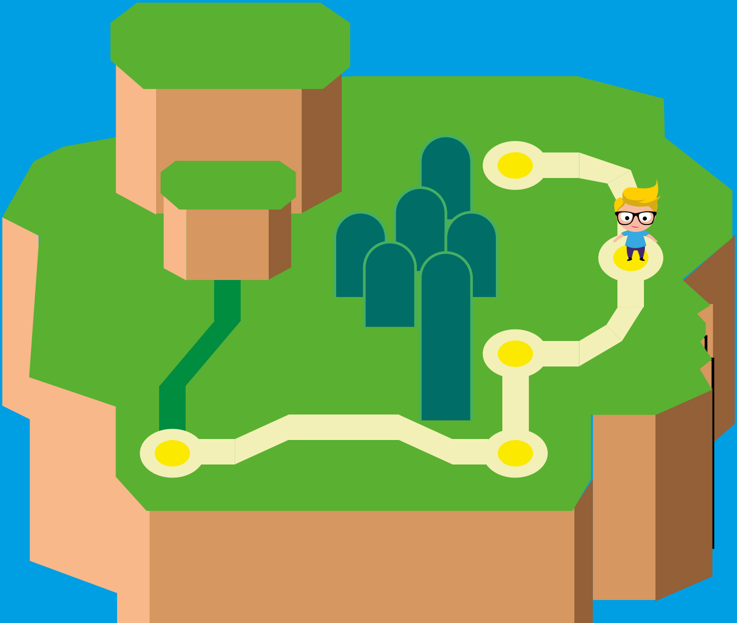
Lean More

- **Introductory article** comparing IoT protocols: *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 ODI Summit 2015: *youtube.com/watch?v=MP2HLCNPgJ0*
- **Eclipse IoT protocols**: *iot.eclipse.org/protocols.html*
- **Realtime data with MQTT**, video covering MQTT and IoT topics: *youtube.com/watch?v=gj8mcn9oWgE*
- **IoT Demonstration using WebSockets**: *developer.mbed.org/cookbook/Internet-of-Things-Demonstration*
- **Vision Mobile report on the IIoT landscape**: *visionmobile.com/product/commerce-of-things-2015*



To end this chapter in a progressive manner, here are some good starting points, representing some of the stakeholder of the industry, software, hardware, aggregators and service providers.

- **Estimotes Blog**: *estimote.com*, Manufacturer of iBeacons and mobile SDK
- **Evothings Studio examples and templates**: *evothings.com/developer*
- **IBMs IOT Foundation**: *internetofthings.ibmcloud.com*, IoT cloudware & apps
- **IFTTT**: *ifttt.com*, If-This-Then-That - a 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 IoT**: *parse.com/products*, Back-end company with lots of client code & libraries, running on top of Amazon
- **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.



BY Robert Virkus

Apps for Wearables

After pioneering work of Metawatch, Pebble and many more companies, Google released Android Wear in spring 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. Apple also unleashed its range of Apple Watches at the end of 2014.

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. Many wearables 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 timepieces 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.

While there are sobering statistics about declining usage of wearables after they have been purchased, wearables as category have firmly established themselves. 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!

Smartwatches

Designing UX for Smartwatches

Whatever platform you might choose: 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 is 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 practices[1]. The guide includes navigation, design and interaction patterns.
- An in-depth article is available from Nielsen Norman

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

Group[2] which combines a review of the Samsung Galaxy Gear smartwatch with design guidelines for smartphone apps.
- Jonathan Kohl wrote a comprehensive article on designing products for smartwatches and wearables[3]

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. 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 optimise notifications for display and interaction on the smart-watch. 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 hand free interaction with your app.

2 nngroup.com/articles/smartwatch

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

- **Wearable app**: you can create apps that run on the smart-watch 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 fine tune 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 community[4].

Apple Watch

The first range of Apple Watches was released in Q1 2015. Apple Watches come in different sizes and a variety of colour schemes to cater for different tastes (and pocket depths).

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

- **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

4 plus.google.com/communities/113381227473021565406

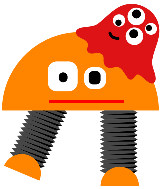
update and interact with the UI that is displayed on the AppleWatch.
- **ClockKit**: Add additional information to the clock face with so called Complications.

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/gear* and *developer.tizen.org*. You can start supporting Tizen smartwatches by sending rich notifications[5] 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.

To keep up with the latest Samsung Gear news follow the Samsung development Twitter channel *@samsung_dev*.



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

Pebble

Pebble is with Metawatch one of the pioneers of the modern smartwatch movement. Pebble nowadays has variety of watches with round and coloured displays.

Your starting point for pebble development is *developer.getpebble.com*. Standalone apps are written in C. You can either use the browser based cloudpebble IDE[6] 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 GitHub[7]. With background apps, access to sensors and AppMessage/AppSync communication options you can create great Pebble apps.

Follow Pebble on Twitter via *@PebbleDev*.

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**, *developer.microsoftband.com*
— **Nike**, *dev.nike.com*

6 cloudpebble.net

7 github.com/PebbleDev/qemu_pebble

- **Polar**, *developer.polar.com*
- **Razer Nabu**, *developer.razerzone.com/nabu*

Smart Glasses

Smart glasses augment the reality with context dependent information - either about what you are looking at or about your current task or both.

Currently smart glasses are mainly used in work scenarios, as their pricing and limited runtime are less attractive for end consumers. One of the best known smart glasses - Google Glass - was shut down and is now being remodelled for business usage. Microsoft's HoloLens was named as one of the top "inventions of the year" by the Time Magazine in 2015.

Here are some smart glasses that are currently in production:

- Epson Moverio[8]
- Google Glass[9]
- Microsoft HoloLens[10]
- Sony Smart Eyeglass[11]
- Vuzix M100[12]
- Westunitis InfoLinker[13]

8 epson.com/cgi-bin/Store/jsp/Landing/moverio_developer-program.do

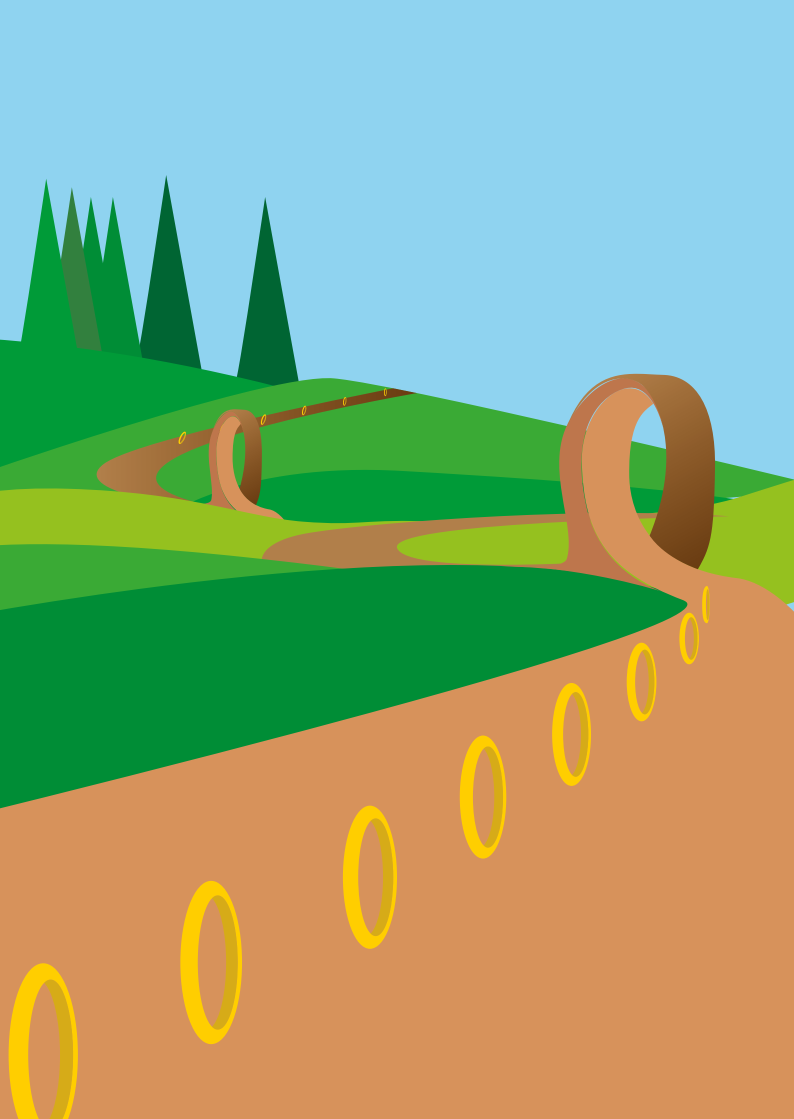
9 developers.google.com/glass/distribute/glass-at-work

10 microsoft.com/microsoft-hololens/developers

11 developer.sony.com/develop/wearables/smarteyeglass-sdk

12 vuzix.com/developer

13 westunitis.co.jp/web/global/wearable/infolinker



BY Dean Churchill

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 2015 Kaspersky Lab detected 323,374 new malicious mobile programs[1]. The historical network and endpoint based defences (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

1 securelist.com/analysis/quarterly-malware-reports/72493/it-threat-evolution-in-q3-2015/

to be used with mobile apps. Input needs to be validated for 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 authorisation 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 switch[2] 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 minimise 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

2 gcc.gnu.org/wiki/Visibility

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 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 vulnerable to snooping and injection attacks. Apps can be installed and inspected in emulators or simulators. Network analysers are freely available and able to monitor, intercept, change and redirect network traffic. Some governments monitor electronic communications for censured topics. Protect all communications using HTTPS instead of HTTP. Downloads of javascript libraries from public sources, like map libraries, should use HTTPS, as hackers, using MATE attacks, can inject malicious code into the download if HTTP is used. Downloads of static content, like pictures, from public sources, should use HTTPS, as hackers could replace images with malicious content. 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. Mobile apps should validate that the hostname or common name of the HTTPS server they connect to is the correct, expected one. 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, 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

Minimise 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.

Minimise 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 minimise the number of permissions or features your application needs.

Tools

Protection

Basic Java code renaming can be done using Proguard[3], an open-source tool and Arxan's GuardIT[4].

3 www.proguard.sourceforge.net

4 arxan.com

Two vendors for managed-code (Java and .NET) protection tools are Arxan Technologies[5] and PreEmptive Solutions[6].

The main vendors for native code protection tools and white-box cryptography libraries are Arxan and Irdeto[7].

Main vendors for secure mobile source code scanning are Checkmarx[8] and HP[9].

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 Paros[10]. A standard network sniffing tool available on common platform is Wireshark[11].

De-Compiling

See the Hex Rays de-compiler[12].

5 arxan.com

6 preemptive.com

7 www.irdeto.com

8 checkmarx.com

9 www8.hp.com/us/en/software-solutions/mobile-app-security/index.html

10 sourceforge.net/projects/paros

11 sourceforge.net/projects/wireshark

12 www.hex-rays.com

Learn More

Here are some useful resources and references which may help you:

- Apple provides a general guide to software security[13]. It also includes several links to more detailed topics for their platform.
- Commercial training courses are available for iOS and Android. Lancelot Institute[14] provides secure coding courses covering iOS and Android.
- A free SSL tester is provided by Qualsys Labs[15].
- Extensive free application security guidance and testing tools are provided by OWASP[16], including the OWASP Mobile Security Project[17].
- A free mobile application performance monitoring tool for IOS and Android is the AT&T Application Resource Optimisation tool[18].

13 developer.apple.com/library/mac/navigation/#section=Topics&topic=Security

14 www.lancelotinstitute.com

15 www.ssllabs.com/ssltest

16 www.owasp.org

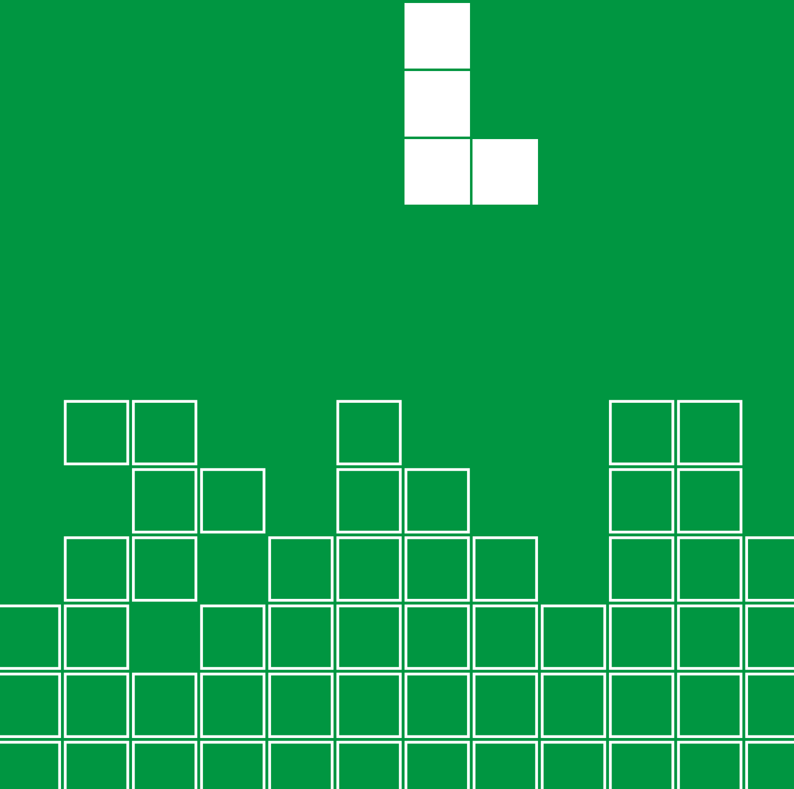
17 www.owasp.org/index.php/OWASP_Mobile_Security_Project

18 developer.att.com/application-resource-optimiser

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. Run a secure code analysis tool against your mobile code to find vulnerabilities. Consider how a hacker would analyse 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.



BY Sally Cain

Accessibility

Why Accessibility is Important

In December 2015 the World Health Organization (WHO) stated on their website that over 15% of the world's population have some form of disability[1] 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. The WebAIM Screen Reader Survey[2] 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.

For example, if they are unable to get out of the house to do their shopping or banking, then providing accessible online services means they can access these services independently. It is important to recognise how important independent access to services is for people with disabilities.

1 www.who.int/mediacentre/factsheets/fs352/en

2 webaim.org/projects/screenreadersurvey6/

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. Embedding accessibility into your apps ensures an excellent user experience for all.
- 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. You may discover a significant new market when you develop apps that suit these users.
- 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.
- 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.
- The organisation 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 devices. 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 robust enough Assistive Technology built into the operating system 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 colour contrast too. iOS, Android, BlackBerry and Windows offer various options to change these in the Settings. As well as the universal 'pinch to zoom' feature, iOS, Android, Blackberry and Windows 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 delivered the Narrator screen reader in Windows Phone 8.1, but it is currently not at the point where it can be used to fully access the phone if you are a blind user. 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 teletype (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 colour 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.

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 organisations 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 Guidelines[5] 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

3 www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39080

4 www.rnib-business.org.uk

5 www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile_access.shtml

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
- 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.

Colours and Fonts

- Ensure there is a good contrast between background and foreground colours. In particular consider buttons which include text. Does the contrast between the text and the background colour meet the ratio requirements in WCAG 2.0 or ISO 9241-171?
- Avoid using colour as the only means of differentiating an action. A colour-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. They should be announced by the screen reader and clearly visible, ensuring they do not disappear from the screen after a short time period.
- Ensure that error messages, notifications and alerts are not provided by colour/haptic/audible output alone. For example, someone with hearing loss will not recognise audible notifications.

Testing

- Do not forget to test your app on the target device with the assistive technology built-in to the OS with more than just the latest OS version. When testing on an Android device please remember that unless the user has a pure Android device, like a Google Nexus, they are unlikely to get access to all of the latest OS upgrades. This is because for OS upgrades you are at the mercy of your phone manufacturer, so the Android OS versions in the wild can be quite diverse. Because some handset manufacturers skin the OS, this can sometimes interfere with the way the accessibility features should work. Therefore it is always recommended that testing for Android Accessibility is undertaken on a Google Nexus device. That way you can be sure there is nothing interfering with the way the accessibility features should work and you are working to a common denominator.
- 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 I/O 2015 presentation[6] and the Google I/O 2013 presentation[7]

The Apple developer program has helpful resources too. Take a look at their accessibility video presentations from WWDC 2014 and 2015 available in the iOS Developer Center[8].

6 youtube.com/watch?v=euEsfNR5Zw4

7 youtube.com/watch?v=ld7kZRpMGb8

8 developer.apple.com/wwdc/videos

Android App Accessibility

The latest major version of Android, Version 6 (Marshmallow), has continued to improve the accessibility support with a new Voice Assistant accessibility API which builds on the work of the Talkback screen reader. Accessibility was really first a realistic proposition with Android 4.1 (Jellybean) and it is much improved since then.

Accessibility features in Android include (but are not limited to) things such as:

- **Voice Assistant (Talkback)** - speech output for blind users
- **Font Size** - for partially sighted users and some users with learning difficulties
- **Magnification gestures** - zoom style magnification for partially sighted users
- **Negative colours** - for partially sighted users and some users with learning disabilities who prefer an inverted colour palette
- **Colour adjustment** - for users who have particular colour preferences
- **Sound detector and flash notification** - for those with a hearing impairment
- **Subtitles** - providing captions or subtitles for those with hearing loss
- **Earphone adjustment and mono audio** - for those with hearing loss using headphones
- **Universal switch** - for those with physical disabilities who prefer to access apps using a hardware device
- **Assistant menu** - to enable those with dexterity issues to change the touchpad or cursor speed and size
- **Tap and hold delay** - for users with motor control issues

There are some helpful resources in the Support Library[9] 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 Accessible[10].

You will also find more examples in the training area of the developer documentation in a section entitled Implementing Accessibility[11]. Testing the Accessibility is also covered online[12].

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. Blackberry have also started to move into the Android space now with 'PRIV'. Please refer to the Android section for information about accessibility features.

The Blackberry Screen Reader[13] is only available for a very limited number of Blackberry Curve devices. It comes pre-installed on the Blackberry 10 devices and can be downloaded

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 mobileapps.blackberry.com/devicesoftware/entry.do?code=bsr

for other supported devices. You can find out which are currently supported on the Blackberry Accessibility Website[14]

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 colours 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 in a dedicated article on BlackBerry's website.

14 us.blackberry.com/legal/accessibility.html

If you are designing for Blackberry 10 there are also developer resources that include some design guidelines[15].

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 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 colours 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
- **Video description** - for people with sight loss.
- **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

15 developer.blackberry.com/devzone/design/bb10/accessibility.html

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 guidelines[16].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 Voiceover[17]

Windows App Accessibility

It is fair to say that Microsoft have been playing catch up with iOS and Android as far as accessibility features go. There has been good support for magnification, text enlargement and changing of colours for some time and in Windows Phone 8.1 things moved on again. There is a screen reader called Narrator in Windows Phone 8.1. 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

16 developer.apple.com/library/ios/documentation/UserExperience/Conceptual/iPhoneAccessibility/Introduction/Introduction.html

17 developer.apple.com/library/ios/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityonYourDevicewithVoiceOver.html

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. However, this is certainly one to watch for the future!

Some of the accessibility features on Windows Phone 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, colour 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.

Accessibility

- **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.

Find out more about Accessibility on Windows Phone[18] including Narrator and other features.

The Accessibility for Windows Runtime Apps documentation[19] provides support whether your are developing in C#/VB/C++ and XAML or JavaScript and HTML.

Microsoft has Guidelines for Designing Accessible Apps[20] 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)[21] which is helpful for making more dynamic content accessible to screen readers.

Once you have tested the accessibility of your app[22], Microsoft uniquely allows you to declare your app as accessible[23] in the Windows store, allowing it to be discovered by those who are filtering for accessibility in their searches.

18 www.windowsphone.com/en-gb/how-to/wp8/settings-and-personalization/accessibility-on-my-phone

19 msdn.microsoft.com/en-us/library/windows/apps/xaml/dn263101.aspx

20 msdn.microsoft.com/en-us/library/windows/apps/hh700407.aspx

21 www.w3.org/TR/wai-aria/markup

22 msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994937.aspx

23 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. 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 Mobile[24]

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)[25] a helpful resource.

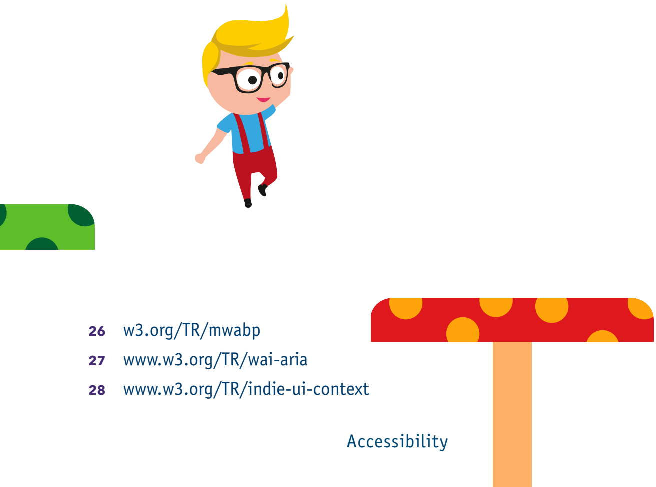
If your app is intended to mimic a native app look and feel, then you should follow the guidelines mentioned above in this chapter.

24 www.w3.org/Mobile/mobile-web-app-state

25 w3.org/TR/mwbp-wcag/

As support of HTML 5 is increasingly adopted on the various mobile platforms, consider reading Mobile Web Application Best Practices[26] as this is likely to form the foundation of any 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)[27], 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 Group[28]. 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 available.



26 w3.org/TR/mwabp

27 www.w3.org/TR/wai-aria

28 www.w3.org/TR/indie-ui-context

BY Julian Harty & Marc van't Veer

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 amphitheatres 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 devices[1]. 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 Wisconsin[2]. A book is also available from HP Enterprises on the confluence between mobile analytics and testing for mobile apps, details are available at *themobileanalyticsplaybook.com*. (co-written by Julian Harty, one of the authors of this chapter.)

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

Testing Needs Time - You Need a Strategy

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

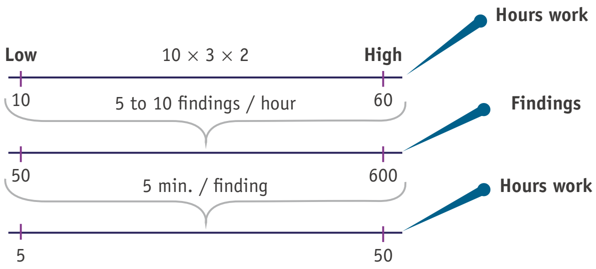
1 blog.testmunk.com/quizup-mastering-android-device-fragmentation-automated-testing/

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

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

Calculation Example

Let us assume your app has ten epics (also called user stories in this context) and every epic needs to be tested using between one and three perspectives (functionality, usability, user scenarios) at a rate of one test per hour. If you test each epic with one perspective and one test case it takes ten hours to test the app. On the other hand, if you test every epic using all three perspectives with two test cases testing takes 60 hours. Typically, testing finds defects and other work worth reporting, which takes more time. You then choose to spend time addressing some of the findings, for instance, to fix a defect or otherwise improve the app. Let us assume that every executed test case results in 5 to 10 findings. The time needed to process the defects lies between 5 and 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 befit ongoing, optimised 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. Commercial tools, such as HP Sprinter[3] can record actions and screenshots to reduce the time and effort needed to report and reproduce problems.
- **Risk Analysis**: You can use the risk analysis to decide how and when to allocate testing effort. Risks are hard to determine accurately by the tester or developer alone; a joint effort from all the stakeholders of the mobile app can help to improve the risk analysis. Sometimes, the mobile app tester is the facilitator in getting the product risk analysis in place.
- **Scaling Testing**: Increasing the throughput of testing by scaling it, for instance using test automation, cloud-based test systems, and more humans involved in the testing can help increase the volume, and potentially the quality, of the testing. Using static analysis tools to review code and other artefacts can also help the team to find and fix problems before the app is released.

3 hp.com/go/sprinter

Involve End-Users in your Testing

Development teams need a mirror to develop a useful mobile app. Early user feedback can provide that mirror. You do not need many end-users to have good feedback[4]. Bigger value is gained with early involvement, multiple users, regular sessions, and multiple smaller tests. Testers can guide and facilitate the end-user testing, for instance, by preparing the tests, processing log files and analysing results. They can also retest fixes to the app.

Whenever others are involved in testing an app, they need ways to access and use the app. Web apps can be hosted online, perhaps protected using: passwords, hard-to-guess URLs, and other techniques. Installable apps need at least one way to be installed, for instance using a corporate app store or specialist deployment services. Possible sources of end-users can be Crowdsourcing[5].

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

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,

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

5 service providers include www.applause.com, PassBrains.com, and TestBirds.de

others apply when creating your tests, and others still when you perform your testing. Testdroid offers a good checklist[6] 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.

- **I SLICED UP FUN**[7]: **I**nput (Test the application changing its orientation (horizontal/vertical) and trying out all the inputs including keyboard, gestures etc.), **S**tore (Use appstore guidelines as a source for testing ideas), **L**ocation (Test on the move and check for localisation issues), **I**nteraction/Interruption (See how your app interacts with other programs, particularly built-in, native apps), **C**ommunication (Observe your app's behaviour when receiving calls, e-mails, etc.), **E**rgonomics (Search for problem areas in interaction, e.g. small fonts), **D**ata (Test handling of special characters, different languages, external media feeds, large files of different formats, notifications), **U**sability (Look for any user actions that are awkward, confusing, or slow), **P**latform (Test on different OS versions), **F**unction (Verify that all features are implemented and that they work the way they are supposed to), **U**ser Scenarios (Create testing scenarios for concrete types of users), **N**etwork (Test under different and changing network conditions)

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

7 kohl.ca/articles/ISLICEDUPFUN.pdf

- **COP FLUNG GUN**[8] summarizes similar aspects under **C**ommunication, **O**rientation, **P**latform, **F**unction, **L**ocation, **U**ser Scenarios, **N**etwork, **G**estures, **G**uidelines, **U**pdates, **N**otifications.

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). Use unique, clear identifiers for key UI elements. If you keep identifiers unchanged your automated 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 'optimise' their mobile code into monolithic blobs of code, however the current devices and mobile platforms mean this form of 'optimisation' is unnecessary and possibly counterproductive. These two topics are both covered in a useful article on the Google Testing Blog, Android UI Automated Testing[9].

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.

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

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

9 googletesting.blogspot.co.uk/2015/03/android-ui-automated-testing.html

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..."[10]. With the combination of different tours in different perspectives (see the I SLICED UP FUN heuristics) coverage and test depth can be chosen.

Examples of Tours[11] 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.

Personas

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

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

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

Testing on Various Devices

Some bugs are universal and can be discovered on any mobile device. Others, and there are plenty of them, are exposed on a subset of devices, and some belong to specific devices. An example of device specific problems with Android on Samsung is *verybadalloc.com/android/2015/12/19/special-place-for-samsung-in-android-hell/*. All this means we need devices to test on and to test on various devices.

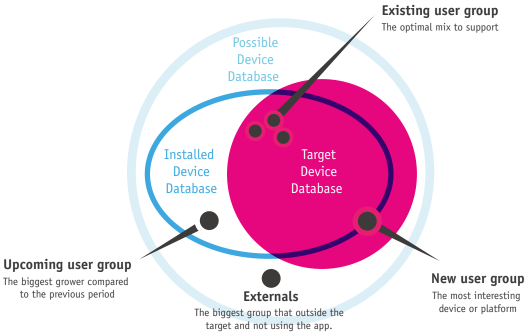
Physical and Virtual Devices

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 include: performance, security, and how we interact with them compared to physical devices. These differences may affect the validity of some test results. Beside the android platform virtual devices you can use *GenyMotion.com*, a faster and more capable Android emulator, for instance, to control sensor values.

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

Ultimately your software needs to run on real, physical, phones, as used by your intended users. The performance characteristics of various phone models vary tremendously from


Existing user group
The optimal mix to support
Possible Device Database
Installed Device Database
Target Device Database
Upcoming user group
The biggest grower compared to the previous period
Externals
The biggest group that outside the target and not using the app.
New user group
The most interesting device or platform

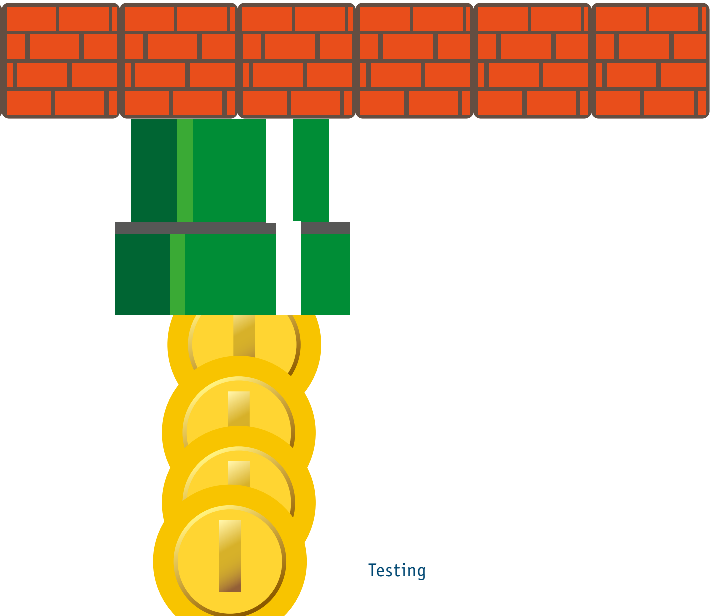
each other, and from virtual devices on your computer. So: buy, beg, borrow phones to test on. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard, screen resolution, networking chipset, et cetera. Try your 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. Samsung[12] (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.

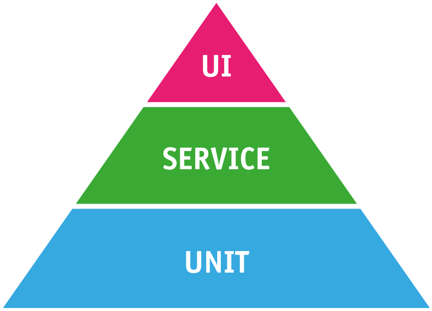
Test Automation

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. Good automated tests mimic good software development practices, for instance using Design Patterns[13], modularity, performing code reviews, et cetera. To automate

12 developer.samsung.com/remotetestlab/rtlDeviceList.action

13 en.wikipedia.org/wiki/Design_Patterns

scripting and coding skills are needed. The level of skills is dependent on the chosen tool. Test automation tools provided as part of the development SDK are worth considering. They are generally free, inherently available for the particular platform, and are supported by massive companies. Test automation can be performed at different levels, see the automation pyramid figure below. It is a strategic choice what should be automated in the unit tests, what on the service or API level and what scenarios on the UI level of the application. The pyramid represents trust that is buildup from the unit test to the higher levels. Multiple test levels are needed to prove that the app works.



GUI Level Test Automation

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

For the tests to be effective in the longer term, and as the app changes, developers need to design, implement and support the labels and other hooks used by the automated GUI tests. Both Apple, with UI Automation[14], and more recently Android[15] use the Accessibility label assigned to UI elements as the de-facto interface for UI automation.

Some commercial companies have open sourced their tools, e.g. SauceLabs' appium[16] and Xamarin's Calabash[17]. These tools aim to provide cross-platform support, particularly for Android and iOS. Other successful open source frameworks include Robotium[18] which now offers a commercial product - a test recorder. Several other tools have effectively disappeared, perhaps the industry is now maturing where the only the stronger offerings survive?

Service Level Test Automation

There is a lot of business logic implemented inside an API. Changes in this logic or in the backend system can be monitored with automated API tests. The focus of the test can be functional-regression but also reliability, performance and security. For functional regression testing a tool like Postman is useful[19].

Several tools can help with API testing. They include Fiddler

14 developer.apple.com/library/tvos/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/UIAutomation.html

15 developer.android.com/tools/testing/testing_ui.html

16 github.com/appium/appium

17 github.com/calabash

18 github.com/robotiumtech/robotium

19 blog.getpostman.com/2014/03/07/writing-automated-tests-for-apis-using-postman/

by Telerik[20]and Charles[21]. Both enable you to view and modify network traffic between your mobile device and the network.

Unit Level Test Automation

Unit testing involves writing automated tests that test small chunks of code, typically only a few lines of source code. Generally, they should be written by the same developer who writes the source code for the app as they reflect how those individual chunks are expected to behave. Unit tests have a long pedigree in software development, where JUnit[22] has spawned similar frameworks for virtually all of the programming languages used to develop mobile apps.

BDD Test Automation

BDD is a Behavior-Driven Development approach with the Test Driven Development family[23]. The behaviour is described in formatted text files that can be run as automated tests. The format of the tests are intended to be readable and understandable by anyone involved with the software project. They can be written in virtually any human language, for instance Japanese[24], and they use a consistent, simple structure with statements such as **Given, When, Then** to structure the test scripts.

The primary BDD framework to test mobile apps is Calabash for Android and iOS[25]. Various others have not been developed

20 telerik.com/fiddler

21 Proxycharlesproxy.com/documentation/getting-started/

22 en.wikipedia.org/wiki/JUnit

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

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

25 github.com/calabash

or maintained in the last year and can be considered defunct for all but the most persistent developers. General purpose BDD frameworks may still be relevant when they are integrated with frameworks, such as appium, that use the WebDriver wire protocol (a W3 standard)[26].

Automation can also help the manual testing, for instance, to replace manual, error-prone steps when testing, or to reduce the time and effort needed, for instance, to automate a collection of screenshots. Developers can help testers to be more efficient by providing automated tools, e.g. app deployment via ADB[27].

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 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 device. 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.

Improvement Cycles

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

26 w3.org/TR/webdriver/

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

the first phase they will be easier to implement in subsequent phases. The five phases might suggest that they follow one after the other and form a logical flow of water down the river. This is not the case. Every step in the different phases provides the possibility to learn and improve. When testing the team learns both how good the mobile app product is and also about areas for improvement in how the app is produced. Mobile app development is a challenging complex, dynamic activity that does not go perfectly the first time, therefore, teams should incorporate an improvement cycle so they can learn and actively improve what they do.

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

Question the design. You want to make sure it fulfills the intended purposes; you also want to avoid making serious mistakes. Phillip Armour's paper on five orders of ignorance[28] is a great resource to help structure your approach.

Phase 2: Verification

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

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) need phones to be configured so development apps can be installed. Based on your test strategy you can decide on which phones, platforms, versions, resolutions are in scope of testing and support.

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

- Usability, user experience and aesthetics requirements
- Performance, particularly as perceived by end users[29]
- Internationalisation and localisation testing

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

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

Phase 3: Launch

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

Given these constraints, it is worth extending your testing to include pre-publication checks and beta tests of the app such as whether it is suitable for the set of targeted devices and end-users. The providers of the main platforms 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. The guideline can be used as a checklist during the implementation phase.

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

Phase 5: Validation

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

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

- Problem detection and reporting. These may include your own code, third-party utilities, and online services.
- Mobile Analytics. Does the data being collected make sense? What anomalies are there in the reported data? What is the latency in getting the results, et cetera?

Learn More

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

- *blog.testmunk.com* Testmunk's blog has a wide range of relevant articles on testing mobile apps.
- *enjoytesting.files.wordpress.com/2013/10/mobile_testing_ready_reckoner.pdf* contains short, clear testing ideas with examples, mainly for Android devices.
- *developers.google.com/google-test-automation-conference/2015/presentations* In 2015, the Google Test Automation Conference (GTAC) includes at least 5 presentations related to testing mobile apps, worth watching.
- *handsonmobileapptesting.com/* which links to the book: Hands-on Mobile App Testing, by Daniel Knott. A well-written book on various aspects of testing mobile apps. A sample chapter is available from the web site.
- *testdroid.com/blog*, a fertile blog on various topics including testing mobile apps. They also have a series on testing mobile games[30].
- *appqualitybook.com*, the website about Jason Arbon's interesting book based on the experiences of testing and analysing vast numbers of mobile apps.
- *appqualityalliance.org/resources,* the official App Quality Alliance AQuA website including their useful app testing guidelines.

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

O

BY Julian Harty

Mobile Analytics

Would you like to know more about how your app is being used and how well it is performing? If so, mobile analytics may be your friend and help you understand the ways your app is being used in-the-wild, by real users. Mobile analytics dovetails with other sources of information, including app store ratings, crash analytics, crowd testing, and usability studies.

Data from mobile analytics can help many aspects of our work, including the business, social, operational, and technological aspects. The data captured can be used to target your work and reduce inefficiencies. You would be in good company, the vast majority of the top 500 iOS and top 500 Android apps include at least one mobile analytics library based on data from SourceDNA[1].

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. Researchers discovered battery drain varied by 3x when their app was used on devices with similar hardware specifications and, amongst other things, they discovered an app used custom code to reduce the screen's brightness when running on Kindle Fire's to improve battery life by 40% and significantly increased the session durations as a result.

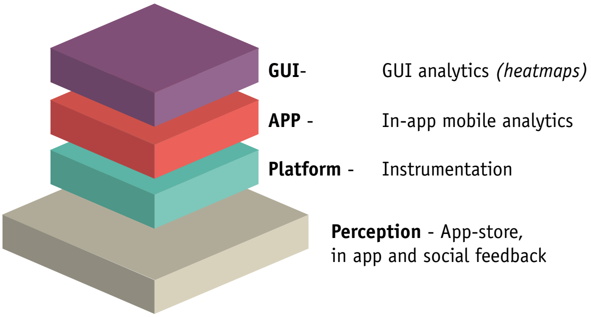
We can also learn ways to improve the ways we develop and test the software. In all the excitement we need to remember to protect user's privacy and respect their preferences and expectations. The effects of mobile analytics can upset users by consuming valuable resources, or abusing sensitive information about the user and their use of the app.

1 sourcedna.com/stats/

Analytics for Each Layer of a Mobile App

Conceptually an app consists of several layers that build on each other. The topmost layer is the UI which communicates with the user. Virtually all apps include a graphic UI (GUI) which is displayed on the screen of the device. There may be other UIs, for instance, to capture movement, audio, and video. The next layer contains the logic, what the app **does**. Often there is also some sort of communications layer. And at the lowest level, there is the physical device with the operating system, or platform, installed, which supports and provides the runtime for the app.

There are various types of analytics available, they can overlap to a certain extent.


GUI- GUI analytics *(heatmaps)*
APP - In-app mobile analytics
Platform - Instrumentation
Perception - App-store, in app and social feedback

The most popular form of GUI analytics is based on heatmaps, they are particularly well suited to capturing data on how the GUI is being used. Heatmaps are enabled by incorporating software into an app to track all the user-interactions with the app's GUI. There are tens of commercial options available, Appsee provide a particularly polished service and offer many free resources including e-books[2] on heatmaps and related topics.

In isolation, heatmap data can be used to track individual user "journeys" through the GUI. In aggregate, various analytics related to User Experience (UX) can be inferred from the data, including problematic areas of the GUI.

There does not appear to be many equivalent services to capture other forms of input, so you may need to write your own code if you want/need to gather data about other UIs.

In-app mobile analytics suits the application logic, it may also record some details of the layer above - the UI - and the layer below - the device.

Analytics Tools

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 open source implementations of their libraries, possibly to allay fears of how their libraries behave?[3]

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 Flurry[4] and KISSmetrics[5]. You often need to register to use the products as many need configuring with a unique 'key' for your app.

Consider several of the potential solutions before commit-

2 appsee.com/ebooks

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

4 support.flurry.com

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

ting 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. Discover what other apps use and why. 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"[6]. Also, Twitter Answers has grown from nothing to become the market leader in 2015[7], an example of how quickly the market is morphing.

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.io[8] abstracts a wide range of mobile analytics offerings. Their opensource code[9] reduces the effort needed to adapt to different analytics providers. Count.ly[10] provide open source implementations of their server as well as of their client

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

7 sourcedna.com/stats/

8 segment.io

9 github.com/segmentio

10 count.ly

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 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 globalisation 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 time?

Many 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 an encoding of the elements and separators. For instance, they may need to be URL encoded[11] when they are sent as REST messages.

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 online[12]. 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 topic[13].

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

12 support.localytics.com/Integration_Overview

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

Configuring your App

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 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.

Windows Phone 8.1 and 10 use the App Specific Hardware ID (ASHWID)[14].

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

To evaluate the quality of the results, some organisations

14 msdn.microsoft.com/en-us/library/windows/apps/jj553431.aspx and msdn.microsoft.com/EN-US/library/windows/apps/windows.system.profile.hardwareidentification.getpackagespecifictoken.aspx

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 triangulation[15].

KISSmetrics provides practical advice on how to test whether the implementation works[16].

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 or the SDKs 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 agreements which we knew few people read and even fewer understand. Instead, make sure your app and any analytics

15 en.wikipedia.org/wiki/Triangulation_(social_science)

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

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.

SafeDK are a recent startup who focus on the behaviour of SDKs, including mobile analytics. SDKs added to apps can adversely affect the performance, security and reliability of the app in addition to other problems and concerns. SafeDK's blog[17] discuss the concerns and provide advice on how to select SDKs by understanding the behaviours they exhibit.

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.

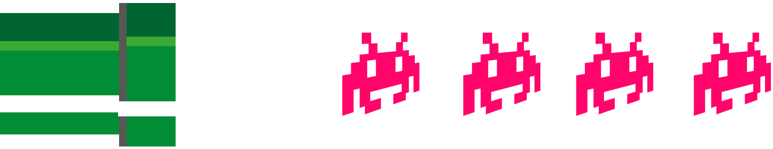
Providers of third-party libraries seem to have a range of attitudes to privacy. Some claim the privacy of users is paramount and stresses the importance of not tracking users.

17 blog.safedk.com/

Google Analytics clearly prohibit tracking personally identifiable information in their terms of service[18]. Others provide examples, including snippets of source code, that demonstrates how to record clearly personally identifiable data. For instance, KISSmetrics provides the following code snippet[19]: `[identify:@"name@email.com"]`. Mixpanel provides an example of how to track specific users[20].

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 analytics: *jnorthrop.me/privacy-considerations-with-mixpanel-people-analytics*
— Kord Davis' book "Ethics of Big Data" (O'Reilly, 2012) available at *shop.oreilly.com/product/0636920021872.do*



18 google.com/analytics/terms/us.html

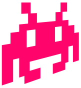
19 support.kissmetrics.com/apis/objective-c

20 mixpanel.com/activity-feed/

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:

- Various articles by Michael Wu or Lithium Technologies. A good place to start is the article "Are Your Big Data Analytics Actionable?"[21]
- Capturing Mobile Experience in the Wild: A Tale of Two Apps[22], 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 Analytics[23], available as a free download.
- The Mobile Developer's Guide to the Parallel Universe[24], a sister book to this one, covers mobile analytics from a marketing perspective.

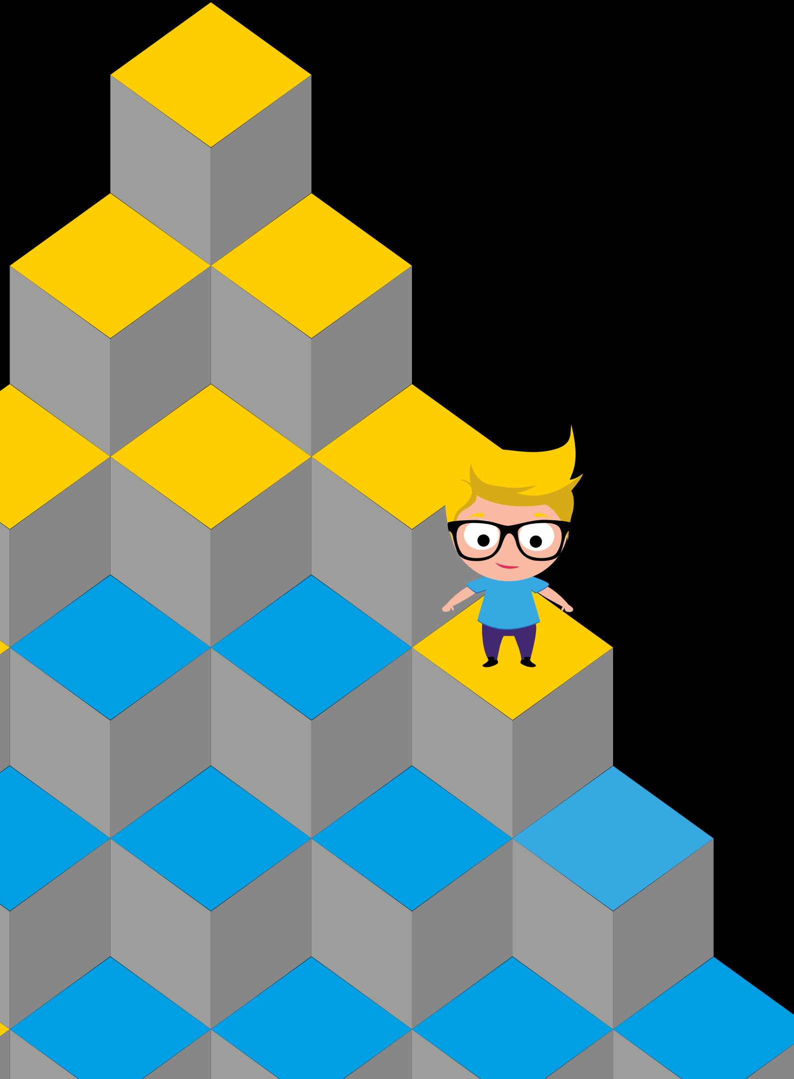


21 community.lithium.com/t5/Science-of-Social-blog/Are-Your-Big-Data-Analytics-Actionable/ba-p/129029

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



BY Julian Harty

Collecting & Understanding User Feedback

With mobile apps, many users willingly provide feedback as app stores can confirm, with some apps receiving 1,000's of distinct feedback per day. App stores have shifted the balance of power subtly to end users who now have a well-known and visible public forums where they can air their gripes and concerns as well as any praise. Their feedback can help us identify potential problems quickly, where they can be "nipped in the bud" i.e. addressed quickly before the problem conflagrates. Furthermore, feedback may include lots of potentially relevant suggestions and recommendations that can be used to improve the mobile app.

Feedback can complement mobile analytics. Mobile analytics provides software-oriented feedback, see the dedicated chapter in this guide to learn more about your options in that area. This chapter focuses on feedback created by humans.

App Store Ratings and Feedback

App stores provide a public forum for users to rate apps and provide written comments. As many of us know, apps with low ratings are much less likely to be downloaded by users. Furthermore, app stores seem to use the app rating as a factor for deciding the priority of including an app in search results and even promoting some of the more popular apps. An excellent read on this topic is the "App Quality Book" by Jason Arbon[1].

1 appqualitybook.com

Ways to Collect Feedback

Various commercial services offer to reduce the effort of collecting and analysing app store data. Examples include appannie.com and appfigures.com. Useful descriptions of roll-your-own analysis are covered in a pair of related blog posts[2]. There are various complications and limitations to collecting feedback from specific app stores such as limits on the number of results available. In many cases you can devise ways to collect the reviews, alternatively a hybrid option is to use services such as AppFigures[3].

Dealing with Inconsistencies

In some instances, the user rating and the remarks may contradict each other. People may not understand the scoring system, where app stores consider 5 stars the highest rating. Users may assume 1 star is the highest rating instead. Alternatively, the ratings may seem to be almost random.

Initially, we might consider the priority of the star ratings on a linear scale, where 1-star is the lowest and 'worst' rating. However, work by Spotify and others discovered that users gave 2-star ratings for the most serious and important problems. They published their findings in a paper called What Do Mobile App Users Complain About[4]?

2 blog.scottlogic.com/2014/03/20/app-store-analysis.html and shinobicontrols.com/news/a-statistical-comparison-of-the-ios-and-android-stores

3 docs.appfigures.com/api/reference/v2/reviews

4 doi.ieeecomputersociety.org/10.1109/MS.2014.50

Rogue Feedback

Rogue feedback is feedback deliberately submitted to affect the overall rating of an app. The feedback may be aimed at inflating or deflating the rating. Some people try to inflate the ratings for various reasons, for instance, to try and get their app promoted by others. Others may target the apps of competitors to drag down the ratings and adversely affect their attractiveness, reduce the number of downloads, etc.

Like SPAM email, some rogue feedback may be easy to detect and either report or filter out from our analysis, for instance if there are lots of identical reviews with the same text, etc. Others may be very poorly written. Some app stores are working hard to reduce rogue feedback. Analogously Amazon has removed lots of fake feedback and is suing various people who wrote the feedback[5].

Social Media

Social media includes services such as Facebook and Twitter, and more recently social video sites such as YouTube, where people share their thoughts, impressions and feelings online with various social groups such as friends, colleagues and acquaintances. They may share things publicly, where anyone can view the shared materials. Some feedback relates to mobile apps. A good example is Facebook's own iOS app which drained the battery of mobile apps. One of the Facebook engineering managers, Ari Grant, explains the causes and the fixes in an online article[6]. Interestingly, some of the subsequent comments indicate the problem may have returned several releases later.

5 www.wired.co.uk/news/archive/2015-10/19/amazon-fake-reviews-legal-action-fiverr

6 facebook.com/arig/posts/10105815276466163

Interpreting and Inferring

As we know from personal experience, the words people write are not necessarily what they mean or exactly what they want to express. They may not spell everything correctly and the grammar may be poor. Also, there are nuances in interpreting writing. For instance, in texting using a period to end a sentence may be considered insincere[7].

Feedback can be in various languages. App stores may use automatic translation to help us read and interpret, none-the-less our understanding will be incomplete and imperfect. Finding native language speakers can help us work more effectively with the users who provide feedback in languages foreign to us. Emotional and sentiment analytics extend feedback in several dimensions, we cover them shortly.

Data Mining

We may need to deal with lots of text on an ongoing basis, for instance, some popular apps receive many 1,000's of pieces of feedback each day, which is expensive to process without software. Data mining can help process vast amounts of data, and help us to identify trends, and discover fresh insights from the feedback we receive. Data mining is a rich research topic, and there is even a dedicated academic research project called UCLappA[8].

There is also a friendly free introduction to data mining written for programmers online[9].

7 lifehacker.com/ending-text-messages-with-periods-can-make-them-seem-in-1747411231

8 www0.cs.ucl.ac.uk/staff/F.Sarro/projects/UCLappA/home.html

9 guidetodatamining.com

Online services such as AppAnalytics from Fido Labs[10] try to automatically process app store feedback, albeit only for iOS apps at the time of writing.

Emotional Analytics

Emotions can be highly relevant for some apps, and the emotions of users can affect their perceptions of an app and any feedback they provide. Apps are available that claim to measure a user's emotions using visual[11] and auditory[12] sources of data. They are early indicators of what might be not only possible but useful, in the near future.

Sentiment Analytics

Sentiment analytics processes what users communicate to determine their feelings, their sentiments, they wish to communicate to others. These others may be us, our organisation, their friends or anyone who discovers what the users have communicated.

Designing for Feedback about the Mobile App

Savvy app developers design their apps and their systems to encourage public feedback when people are more likely to provide positive feedback and direct feedback if the feedback is likely to be negative or critical. Users may be in a good mood after successfully completing an action, for instance, a level in a game, or a purchase on an e-commerce site. A discreet request for feedback on completion may be well

10 apps.fidolabs.com

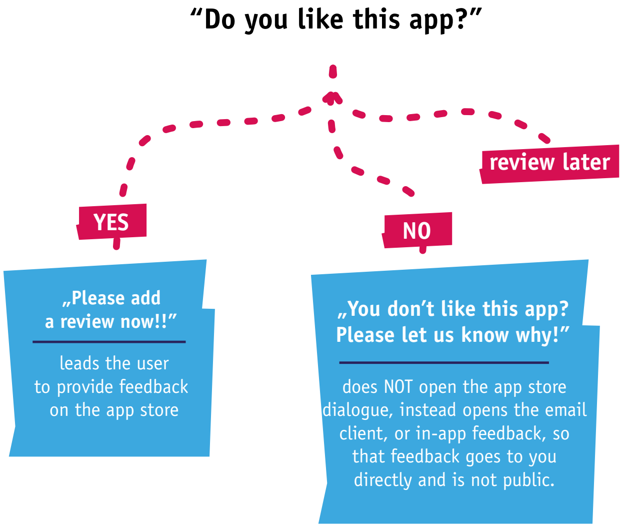
11 affectiva.com/solutions/mobile

12 beyondverbal.com

received by some users, and others can simply continue with whatever else they want to do. Some designers may include prepared feedback statements in the hope of encouraging users to use them.

Asking for feedback in your mobile app is a simple way to get started. However, when designing feedback try not to obstruct or frustrate users, for instance, do not block them midway through a process, a game level, etc. - at least do not if you have any hope of getting positive feedback.

When implementing a feedback mechanism into your app, you may want to route negative feedback away from the app store, where it would be public and adversely affect the app's rating on the store. A dialogue offering the following options would be one way of achieving this:


"Do you like this app?"
YES
NO
review later
„Please add a review now!!"
leads the user to provide feedback on the app store
„You don't like this app? Please let us know why!"
does NOT open the app store dialogue, instead opens the email client, or in-app feedback, so that feedback goes to you directly and is not public.

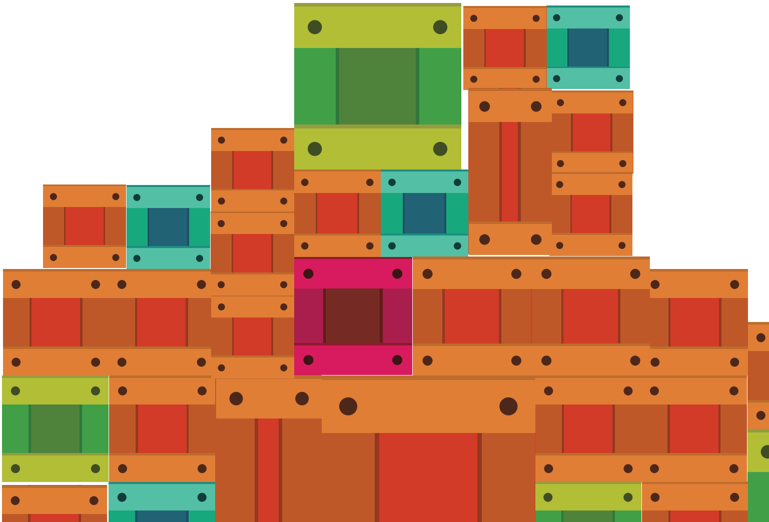
Some project teams may decide to incorporate commercial feedback services, such as HelpShift[13], to help them proactively manage feedback by users of their app.

Responding to Feedback

For many organisations being able to respond accurately and rapidly enables them to improve not only the user's perceptions but also the perceptions of many more people either directly or indirectly (for instance through the user telling others about the good things we have done). None-the-less, according to appbot[14] on Google Play 97% of reviews go unanswered. Perhaps you can out compete many of your competitors by answering all your reviews?

13 helpshift.com

14 blog.appbot.co/97-of-google-play-app-reviews-go-unanswered

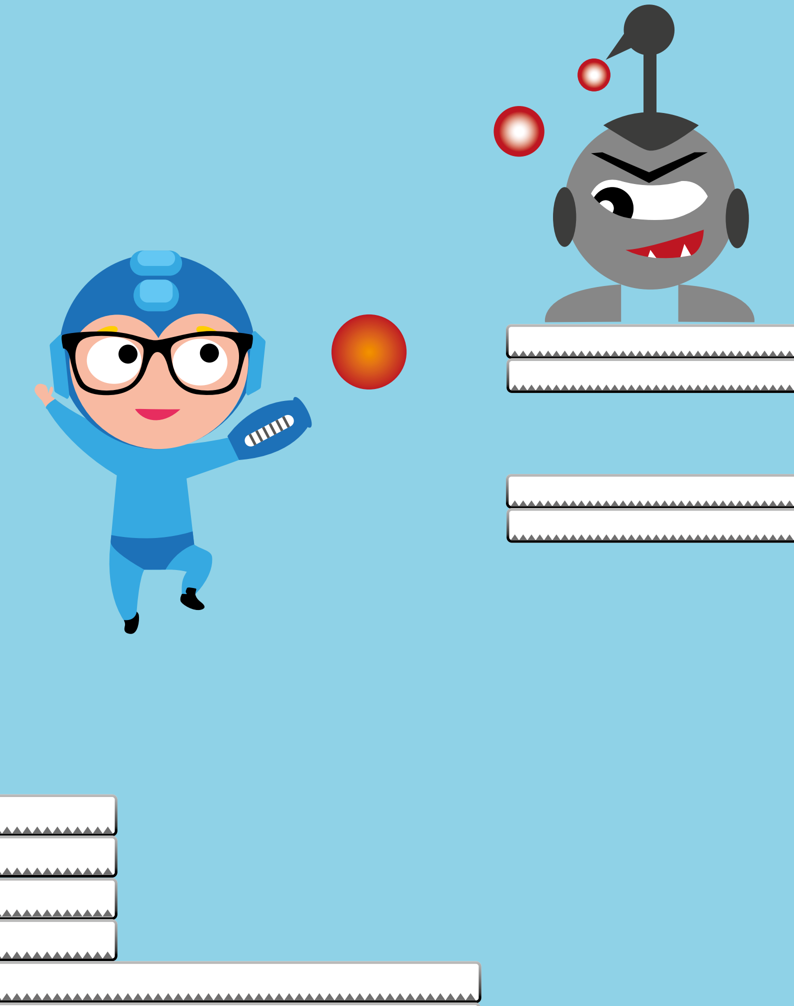


Most app stores offer the opportunity to respond to reviews for their apps. By responding (especially for bad reviews), app providers can help their users and increase engagement. If you help users solve an issue they encountered, they can (and hopefully will) revert their poor rating and give your app more stars than in their initial review. **Caution**: Do not respond with general templates, otherwise you may give users the impression that you are not treating their review with sufficient respect (e.g. "Thanks for your feedback, we will look into this."). Instead, let the user know they are special and their feedback is valuable by incorporate details they provided (including their name if it is available).

A book called The Art of the App Store[15] devotes a chapter to Feedback. Key points include

— Categorizing feedback, for instance, identifying constructive criticism. Vague comments can be filtered out at this stage too.
— Converting feedback into actionable tasks, including suggested fixes and assigning priorities.
— Updating the app with various fixes and improvements.
— And finally, wording the release notes so users can easily read the good news about the improvements and install the updates promptly.

15 ISBN 978-0-470-95278-8



BY Michel Shugair

Monetisation

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 monetisation 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
- **App platform subscriptions:** Create small apps and lease them to businesses

When you come to planning your own development, determining the monetisation business model should be one of the key elements of your early design as it might affect the functional and technical behaviour of the app. Five strategies to monetise your mobile app[1] is an excellent article on how to design the financial aspects so they do not 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 set up a mechanism yourself. Once the most popular and profitable monetisation 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% today[2].

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 monetisation 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. 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).

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 implement a payment mechanism through that, such as PayPal mobile, dial-in to premium landlines[3] 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

3 daopay.com

price elasticity of demand PED[4]. 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 licence” 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 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 monetisation 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.

4 en.wikipedia.org/wiki/Price_elasticity_of_demand

Distimo reported in 2014 that In-app purchases accounted for 79% of iOS revenue[5].

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.

Mobile Advertising

As is common on websites, you could decide to earn money by displaying advertisements. According to VisionMobile's survey among 13,000 developers[6], almost half of mobile app developers are still reliant on this revenue model- although it seems proven that it's proving profitable for only a small minority:

5 "2014 How the Most Successful Apps Monetize Globally" available at www.distimo.com/publications

6 vmob.me/DE3Q15

83% of the study's participants who rely on ads, make less than $10,000 a month.

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 monetises 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 Madgic[7], smaato[8] or inneractive[9] as they tend to bring you better earnings by combining and optimising 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 Apponsor[10] 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 Funder[11].

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 app programs**: Promote third party or your own paid apps within a free app. *MobPartner.com* is a service provider that offers this type of monetisation.
2. **Data reporting**: Track behaviour 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. Only 10% of the participants in the DevEconomics Report Q3 2015 are relying on this business model, although mobile commerce developers are more than three times as likely to make over 100K USD per month than those monetising with ads[12].

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.

12 vmob.me/DE3Q15

Component Marketplace

A Component Marketplace (CMP) provides another opportunity for developers to monetise their products to other developers and earn money by selling software components or white-labelled apps. A software component is a building block piece 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 licence 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 licence. Some licences 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. ChupaMobile[13] is a relevant player in this domain.

App Platform Subscriptions

The 2nd wave after the mobile consumer app adaption came from the small and medium enterprises. According to Gartner[14] the development capacity required to meet the demand of enterprise apps will become critical in 2017.

Smaller businesses, including your bakery around the corner, are interested to have their own app, but do not have the budget to justify the development. For this target market the App Platforms are an excellent choice. The developer designs an app with some default options, adds the content and sells a subscription to the company. All content is hosted online, only the app framework is downloaded from the app stores. So the company has to renew his subscription each month or year to keep his app alive. Mastering the App Platform development tools and selling a couple of new subscriptions a month ensures a recurring revenue for the developer.

Choosing your Monetisation 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

13 www.chupamobile.com

14 www.gartner.com/newsroom/id/3076817

assessment of your app is misplaced or the competition fierce. The Application Developer's Alliance recommends the PPD monetisation method mainly for high-production apps, apps with barriers to entry and high-volume apps[15]. 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.
- 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.

15 www.appdevelopersalliance.org/app-monetisation

App Store 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. App stores 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 app store 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 app stores is that they are distribution channels and not marketing machines. This means that while app stores 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 app stores.

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 app store, 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 app store, make sure you drive traffic to your app through your website, SEO and social media.

Multi-Store vs Single Store

With 120+ app stores available to developers, there are clearly many application distribution options. But the 20 minutes needed on average to submit an app to an app store 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 app stores can have serious limitations, such as payment mechanisms, penetration in certain countries, content guidelines.	90%+ of smartphone users only use a single app store, which tends to be the platform app store shipping with the phone.
Smaller stores give you more visibility options (featured app).	Your own website can bring you more traffic than app stores (especially if you have a well-known brand).
Smaller stores are more social media friendly than large ones.	Many smaller app stores 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 app store.

The platform app stores should give you general coverage for users, but over time, it is in your interest to adapt your app store strategy to match your targeted user base, and utilize the app stores 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 app stores with large audiences, such as the Amazon app store for Android, which offers developers a number of ways to monetise their apps, such as PPD and in-app payments in several countries. Additionally, in some countries, there are locally popular app stores, such as AndroidPit 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. 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, over 50% of them are below the "app poverty line" of $500 per app per month[16].

Mobile games seem to offer the most options to make money according to Pulse[17]. And even if Android phones and tablets are outnumbering iOS devices, revenues generated in Apple's App Store are 80% higher than on Google Play[18].

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.

16 www.vmob.me/DE3Q15

17 www.linkedin.com/pulse/mobile-gaming-monetisation-making-more-money

18 techcrunch.com/2015/10/15/ios-app-store-revenue-now-80-percent-higher-than-google-play-thanks-to-china/

GAME
OVER

Epilogue

Thanks for reading this 16th 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 . If you are using Twitter, follow the project on *twitter.com/mobiledevguide*.

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* where you also find more information about Enough Software, the German app agency who is coordinating this project since it started in 2009.

The 17th edition can be expected for early 2017!

About the Book

This project was initiated by Enough Software in 2009 with the aim to spread knowledge about mobile technologies and to encourage people to enter our community or deepen their existing knowledge. We have given away more than 80,000 hardcopies at events worldwide. Universities and schools in Germany, Netherlands, UK, Spain and South Africa use the book as teaching material. The electronic versions (ebook and pdf) have been downloaded hundreds of thousands of times and the content has been translated into several languages. The book is a non-profit project: the writers, editors, translators and designers contribute their work free of charge. The printing and distribution costs are covered by sponsors.

The Publisher

Enough Software

Enough Software is an app agency from Bremen, North-Western Germany with over 10 years of experience. Our team of 25 experts design and develop applications for any kind of business and for any platform. Besides we are always glad to share our experience - be it with this book, as consultants, or trainers. We value open, transparent communication both within project teams and when working with our clients. Companies like BBC, Nokia, Vodafone, Samsung and CEWE use our tools for their projects or work with us as their application development partner.

www.enough.de

@enoughsoftware

The Printing Sponsor

Microsoft

Microsoft is a technology company whose mission is to empower every person and every organization on the planet to achieve more. Founded in 1975, our strategy is to build best-in-class platforms and productivity services for a mobile-first, cloud-first world. We develop, license, and support a wide range of software products, services, and devices that deliver new opportunities, greater convenience, and enhanced value to people's lives. We offer an array of services, including cloud-based services, to consumers and businesses. We design, manufacture, and sell devices that integrate with our cloud-based services, and we deliver relevant online advertising to a global audience.

www.microsoft.com

The Authors & Contributors

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

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.

@davocbradley

Sally Cain / RNIB

Sally has worked at RNIB in the area of digital accessibility for more than 16 years. She believes passionately in equal access to digital technology for people with disabilities. Sally is her organisation'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 is currently the Accessibility Technology Manager at RNIB with responsibility for the accessibility of all internal and customer facing systems, ensuring that RNIB is delivering on accessibility not only for customers, but for staff too. She also led the writing of RNIB's own internal app standard for accessibility.

www.rnib.org.uk

@sallycain

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.

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.

@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.

www.gamesasaservice.net

@athanateus

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.

@ovvyblabla

www.ovidiuiliescu.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.

www.evothings.com

@dr_alexj

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.

www.enough.de

@linux_pinguin

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

Vikram Kriplaney / local.ch / iPhonso.com

Vikram has been a mobile developer since when WAP was still cool and Symbian and J2ME were still fashionable. He founded mobile at local.ch in 2007, where he went on to singlehandedly develop massively successful mobile web, iOS and Android apps. He's now Mobile Architect and lead engineer for local.ch and search.ch (Swisscom Directories).

He's very Spanish, very Indian and increasingly Swiss – having grown up between Gran Canaria and Mumbai, he calls Zurich home.

Nowadays, you'll find him raving madly about Swift, while nurturing crazy app ideas at iPhonso.

local.ch iPhonso.com

@krips

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

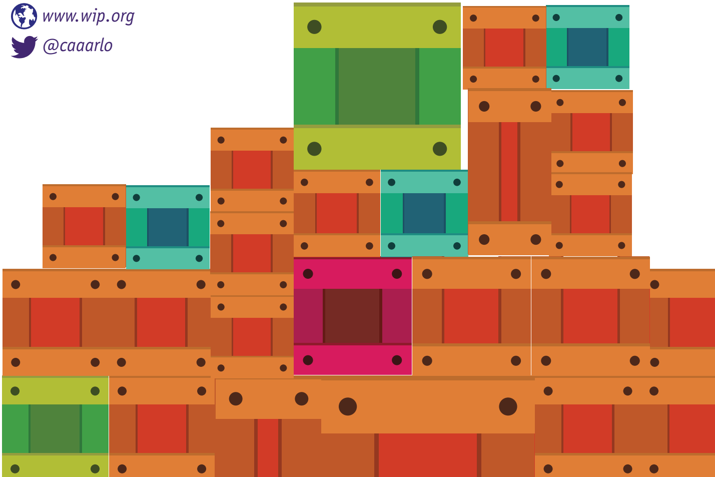
Carlo Longino / WIP

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.

www.wip.org

@caaarlo



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.

www.timmesserschmidt.com

@seraandroid & @PayPalDev

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

Alex Repty

Alex is a freelance software engineer specialising in OS X, iOS, watchOS and tvOS software. He has been developing software for Apple platforms ever since he got his first Mac in 2004. Since then, he has helped create a wide variety of applications, some of which were even featured in Apple's "There's an app for that" campaign or won an Apple Design Award. His passion for clean code, software engineering trends and user experience design make him get up on stage on various iOS-related conferences.

@arepty

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 monetisation 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.

www.mobiledevelopersguide.com & www.enough.de

@enoughmarco

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.

scn.sap.com/blogs/ithain/ & www.sap.com

@ithain

Marc van 't Veer / Polteq

Marc is a mobile app test consultant and trainer at Polteq and has worked in different test roles for over 9 years. He is experienced in testing in a technically oriented context, such as telecom, SOA, test automation and testing API's. Currently Marc supports companies in improving the mobile app testing based on the TI4 Mobile approach.

marcvantveer.niobe.nl & www.polteq.com

@marc_vantveer

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.

www.j2mepolish.org & www.enough.de

@robert_virkus

Mladenka Vrdoljak / Enough Software

Mladenka is completing an apprenticeship as a digital media designer at Enough Software. That means she is engaged with UI, graphic design for mobile applications, as well as coding. She is also responsible for the design of this guide.

www.enough.de

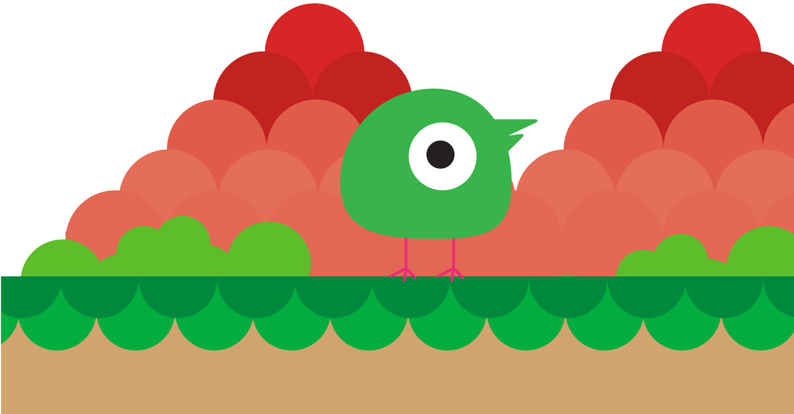
@_mladenka

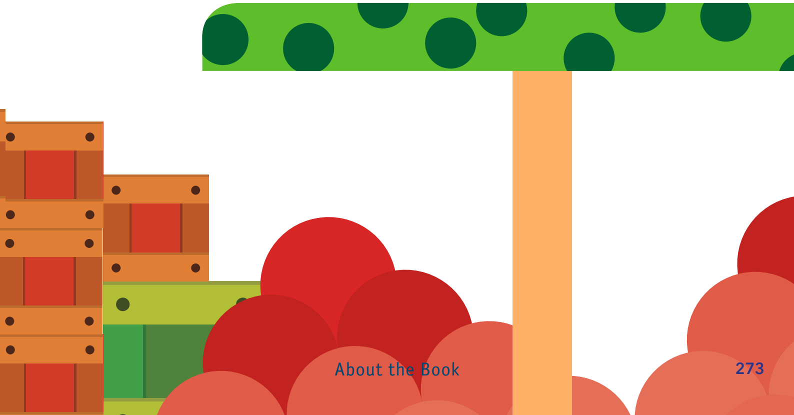
Chris Ward / Sitepoint

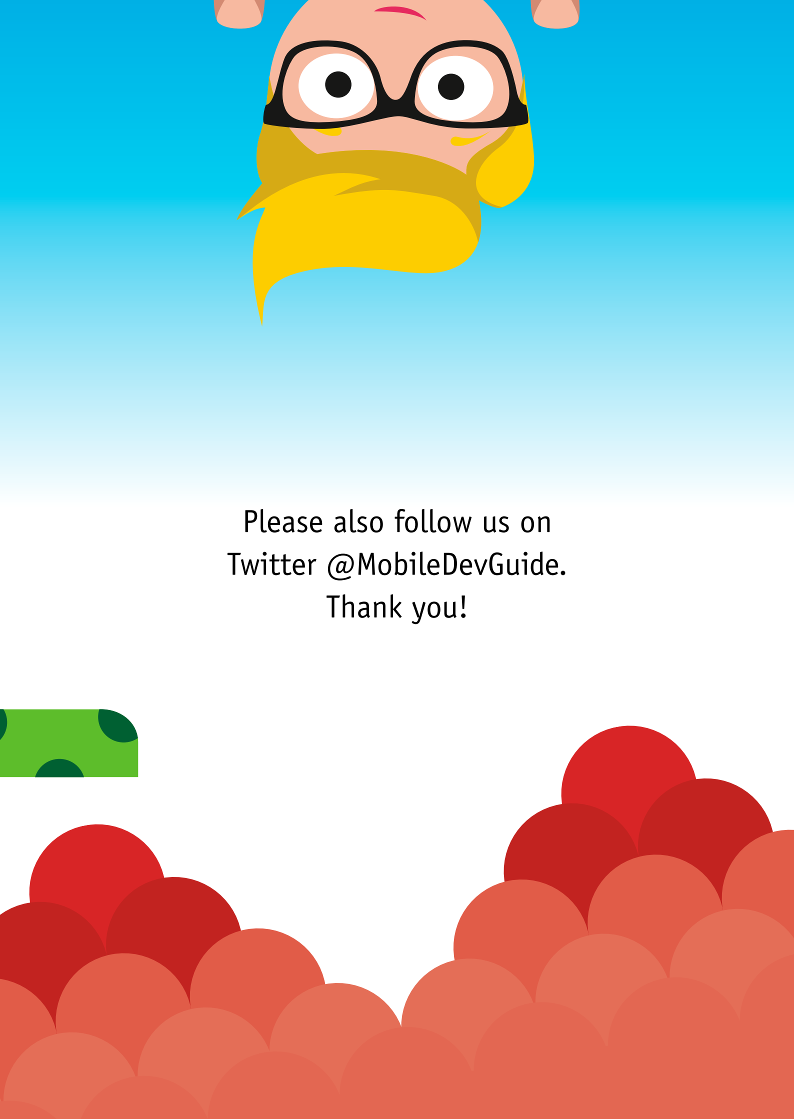
Chris is a globe-trotting developer and writer currently working on several projects that all aim to explore the potential of 'open culture'. Currently the editor of www.sitepoint.com/mobile he is always looking for new writers.

chrischinchilla.com

@ChrisChinch







Please also follow us on
Twitter @MobileDevGuide.
Thank you!