Don't Panic

MOBILE DEVELOPER'S GUIDE TO THE GALAXY

published by:

ENOUGH SOFTWARE

Services and Tools for All Mobile Platforms

Enough Software GmbH + Co. KG
Sögestrasse 70
28195 Bremen
Germany
www.enough.de

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

12th Edition February 2013

Editors:

Marco Tabor
(Enough Software)

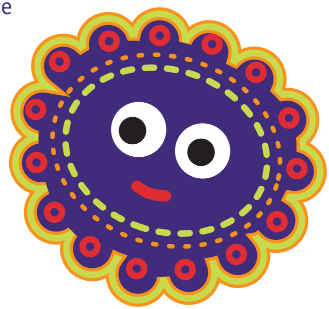
Julian Harty

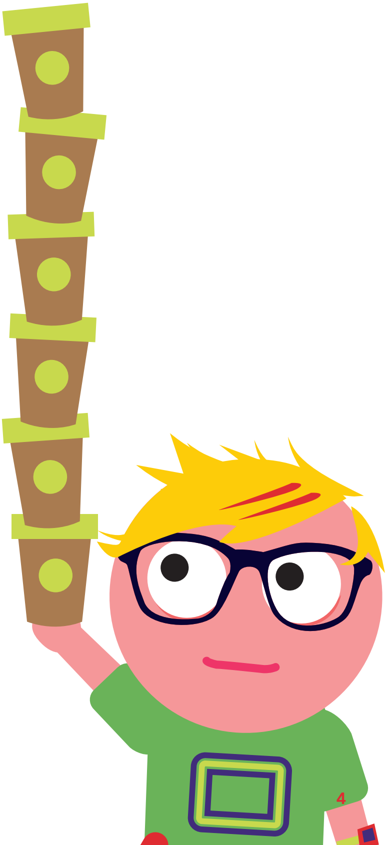
Izabella Balce

Art Direction and Design by

Andrej Balaz
(Enough Software)

Mobile Developer’s Guide
Contents





Prologue

It has been exactly four years now since we published the first version of this guide. In the prologue of that first guide, we said: "The most important issues for mobile developers are for sure fragmentation and distribution". Although the mobile ecosystem's landscape has seen massive changes since then, we still think this holds true. And it keeps changing: This 12th edition probably introduces more changes than any other release before. To reflect recent shifts in the market, we decided to remove the dedicated chapters about Qt and bada and replace them with new, platform-independent chapters which cover key technologies such as Augmented Reality, Analytics and Prototyping. Additionally, the general introduction chapter has been significantly expanded and now provides a comprehensive, updated overview on the ever changing mobile world. All other chapters have, of course, also been revised and updated. You will hardly find a more objective and up-to-date guide that teaches you how to get started as an app provider or developer, and how to lift your existing mobile offering to the next level.

What happened in the mobile industry since our last release? Apple released iPhone 5 with great success, Android became the clear global leader in smartphone systems, and Microsoft released both Windows 8 and Windows Phone 8. Mark Zuckerberg criticized Facebook's own HTML5 approach and found massive usage growth after turning to native apps. Then there was Canonical presenting Ubuntu for mobile phones. Finally, BlackBerry surprised us developers with love songs and released the BlackBerry 10 OS.

We will establish a bi-annual release cycle from now on. So expect the 13th edition around October 2013. If you are missing content, want to get involved one way or the other, or know of potential sponsors for the project: Get in touch via developers@enough.de. This is an open community project and we invite all of you to push it even further forward.

We are looking forward to hearing from you!

Robert + Marco / Enough Software
Bremen, February 2013

BY Robert Virkus & Marco Tabor & Matos Kapetanakis

The Galaxy of Mobile: An Introduction

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

The focus of this book is on developing mobile apps, which includes a number of phases including: planning & specification, prototyping & design, implementation, internal testing & 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, we advise against further delay as the world is moving speedily towards mobile becoming the predominant form of computing and others will surely bypass you if you wait too long.

While developing mobile apps has some commonalities with developing other software it has specific characteristics. We will cover some of these next.

Topology: Form Factors and Usage Patterns

You have to differentiate between smart phones, tablets and feature phones. Each form factor poses its own usability challenges; for instance, a tablet demands different navigation than a phone. TV systems are getting more traction as another form factor for mobile developers.

Android usage patterns, of course, differ from those on iOS, which also differ from those for Windows Phone apps, et cetera.

You should, therefore, refrain from providing the identical experience on all form factors or even all your target smartphone systems. Otherwise, you risk delivering a mediocre service to some sections of your target user base.

Star Formation: Creating a Mobile Service

There are several ways to realize a mobile service:

- App
- Website
- SMS, USSD[1] and STK[2]

App

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

Native Apps

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

1 en.wikipedia.org/wiki/USSD
2 en.wikipedia.org/wiki/SIM_Application_Toolkit

Web Apps

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

A famous example for a web app is the Financial Times app which left the app store in order to keep all subscriber revenue to themselves for the web world; inversely, 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 it so that you can publish them in app stores, e.g. Phonegap[3].

Hybrid Apps

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

For many mobile app developers, this controversy does not really exist any longer as a hybrid approach to app development has become quite common place: 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 same app. A hybrid app makes

3 www.phonegap.com

use of native and web technologies. Parts of it behave as a native app, while other parts are powered by web technologies. These parts can use Internet connectivity to offer up-to-date content. While this could be viewed as a drawback, the use of web technologies enables developers to display up-to-date content without the need to re-submit the application 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 for the most part on your server but you can access various phone features on the device with JavaScript, e.g. to store data locally or to request the current location of the device. In contrast to apps, mobile websites are inherently cross-platform. However you should not assume that a mobile browser is always based on WebKit, made evident by Microsoft's plea to mobile web developers not to make their websites run only on WebKit[4].

SMS, USSD and STK

Simple services can be realized with SMS, USSD or STK. Everyone knows how SMS (Short Message Service) text messaging works and every phone supports SMS, but you need to convince your users to remember textual commands for more complex services. Some operators offer APIs for messaging services that work for WiFi-only devices, such as the network APIs of Deutsche Telekom[5]. USSD (Unstructured Supplementary Service Data) is a GSM protocol used for pushing simple text based menus, the capabilities depend on the carrier and the

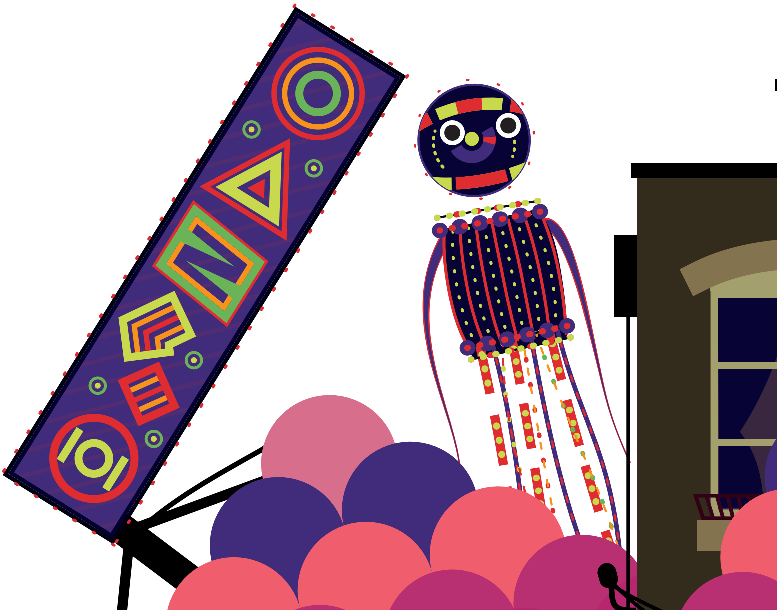
4 blogs.windows.com/windows_phone/b/wpdev/archive/2012/11/15/adapting-your-webkit-optimized-site-for-internet-explorer-10.aspx

5 www.developergarden.com/apis

device. STK (SIM Application Toolkit) allows to implement low-level but interactive apps directly on the SIM card of a phone.

STK may appear irrelevant when we focus on smartphone apps, however m-pesa, for example, is a STK app which is transforming life and financial transactions in Kenya and other countries.[6]

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



The Universe of Mobile Operating Systems

The mobile space is much more diverse than other areas in IT. When you are developing software for stationary computers, you basically have 3 operating systems to chose from. When it comes to mobile, there are many more. This book will give you an introduction to the ones that are currently the most relevant but be aware that 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 disappear again within a short time, no matter how big the companies behind them or their former market relevance might have been.

So read on, learn how the market looks today and keep observing it on your own (or make sure you have the latest edition of our guide at hand).

Quasars: Android and iOS

When people talk about mobile apps, they mostly are only referring to Android and iOS. Why? When it comes to market share, these two platforms combined, dominate the smartphone market with almost 90% in key markets, particularly, the US[7] (see the table below for global numbers). The Developer Economics 2012 research[8] also shows that iOS and Android are at the top in terms of developer mindshare – i.e. the percentage of developers using each platform, irrespective of which platform they consider to be their 'primary'. Android was at the top, with 76% of developers currently working on the platform, followed by iOS with 66%. Both platforms showed an increase in interest compared to 2011.

7 blog.nielsen.com/nielsenwire/online_mobile/nielsen-tops-of-2012-digital/
8 www.DeveloperEconomics.com

Another 2012 research report, by Appcelerator and IDC[9] agrees that developers are mainly interested in Android and iOS, although the figures are reversed. This study indicated that nearly 90% of respondents were interested in iOS, with Android somewhat lower, at 80%. Despite their differences, both these reports point to one fact: if you are going to use Android or iOS, you will have lots of competition.

Dark Matter: Feature Phone Platforms

While smartphones get all the news most parts of the world belong to the feature phone universe. Globally 60% of all phones sold in Q3 2012 have been feature phones[10], with an install base much higher than that. Biggest vendors are Samsung and Nokia. Nokia claims to have quite a lot of success with their Nokia Store as there are more than 500 developers who have had more than 1 million app downloads of their app[11]. Research from 2011 showed that the unhyped platforms actually provided a better chance for developers: Feature phone apps on Nokia's OVI store had 2.5 times higher download numbers compared to apps on Apple App Store[12].

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

9 www.idc.com/getdoc.jsp?containerId=prUS23619612

10 www.gartner.com/it/page.jsp?id=2237315

11 www.developer.nokia.com/Distribute/Statistics.xhtml

12 www.research2guidance.com/apps-on-nokia's-ovi-store-had-2.5-times-higher-download-numbers-in-q2-2011-compared-to-apps-on-apple-app-store/

Super Novas:
Windows 8, Windows Phone, BlackBerry 10 and Aliyun

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

While initial Windows 8 adoption was higher in absolute numbers compared to Windows 7, relative adoption has been slower[13]. PC sales continues to decline, perhaps consumers are waiting for compelling touchscreen models that promote the benefits of Windows 8? Reception of Windows 8 varies between outrightly hostile and positively euphoric.

Windows Phone received some good press coverage with its WP8 release but we still have to wait for actual market numbers. Global adoption seems to be slowly increasing and in Italy, Windows Phone has even been breaking the 10% barrier[14]. Apps have roughly doubled in 2012 and app downloads have increased, so there is still hope.

BlackBerry 10 has just been released, the initial reception varied between skepticism and enthusiasm - but most importantly all relevant operators will carry BlackBerry 10 devices.

Aliyun has been released on a single device in China with an unknown market share. It draw publicity mostly from the fact that Google pressured Acer into not releasing an Aliyun device based on Acer's membership of the Open Handset Alliance and on the fact that Aliyun's app store featured some pirated Google Android apps[15]. While Aliyun is claimed to be based on Linux, the source code has not yet been released.

13 phonearena.com/news/Wait-so-Windows-8-is-not-outpacing-Windows-7-adoption-rate_id37212

14 guardian.co.uk/technology/2012/oct/02/windows-phone-europe-market

15 news.cnet.com/8301-1035_3-57513651-94/alibaba-google-just-plain-wrong-about-our-os

White Dwarfs: Symbian and bada

Only shadows of their former selves are Symbian and Samsung bada. While bada was very shortlived, Samsung announced it will live on within the forthcoming Tizen platform - though they did not specify in which specific form this will happen. Symbian has been pushed into maintenance mode - even though Nokia still releases some hallmark Symbian based devices such as the PureView 808, the importance and market share continue to fall sharply worldwide.

Newborn Stars: Firefox OS, Mer, Sailfish, Tizen, Ubuntu

We will see some interesting new entries in 2013.

Mozilla's Boot to Gecko initiative is now called Firefox OS[16]. This operating system is based on open web technologies and ZTE for which Alcatel will release devices in 2013. Interestingly Firefox OS has already been ported to the popular hacker platforms Raspberry Pi and Pandaboard[17]. Firefox OS will target low entry devices primarily, so it may have a big impact on future smartphone sales in developing countries.

The Mer project[18] continues the MeeGo platform and provides the basis for Jolla's[19] Sailfish OS. Sailfish OS is expected to launch in Q1 2013, its responsive and gesture based UI has been praised by some previewers.

Tizen[20] devices are expected to be released in 2013 by Samsung and Lenovo. Seemingly only gently pushed forward by Samsung and Intel, Tizen aims to power not only smartphones but also TVs, tablets, netbooks and in-vehicle infotainment systems.

16 mozilla.org/firefoxos
17 rawkes.com/articles/there-is-something-magical-about-firefox-os
18 merproject.org
19 jolla.com
20 tizen.org

Last but not least Canonical presented Ubuntu[21] for mobile devices. The idea is to bring the full power of a PC to the phone.

Hard Numbers: OS Market Shares

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

Platform	Market Share end of 2012	Shipments in Q3 2012	Installed base end of 2012
Android (Google)	53%	129 million	710 million
iOS (Apple)	19%	27 million	260 million
Symbian (Nokia)	14%	4 million	190 million
BlackBerry (RIM)	8%	8 million	104 million
bada (Samsung)	>2%	5 million	30 million
Windows Phone (Microsoft)	<2%	4 million	20 million

(Source: http://www.tomiahonen.com/ebook/phonebook.html)

21 ubuntu.com/devices/phone

You may agree with the majority of developers and decide that spending time on platforms other than iOS and Android would be a waste of time. Be assured: It is not that simple. Smartphone owners are still a minority. More people own feature phones and in many regions of the world, people still choose a feature phone over smartphones when purchasing a new device: For instance, in the whole MEA region, 80% of all sold phones in Q4 2012 were feature phones[22]. These devices do not even appear in the table on the left – probably because feature phone users do not frequently use apps and are, therefore, not of interest to many developers.

Newer platforms such as BB10 do not appear in many statistics either because they are new to market. However, one of these platforms might still be the best choice for your business case.

You also have to remember that these are global figures - the regional market share of each platform is another matter altogether. In a world where localized content is increasing in importance, it is essential to know the details and characteristics of your home market. For example, China is the largest smartphone market today where Android clearly dominates, holding a market share of over 90%.[23].

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

22 gulfnews.com/business/technology/apple-microsoft-to-steal-market-share-in-mea-smartphone-sales-1.1112944
23 www.techinasia.com/android-market-share-china-2012/
24 comscoredatamine.com/category/mobile
25 gs.statcounter.com
26 visionmobile.com
27 gartner.com

About Time and Space

As developers, we tend to have a lot of passion for our chosen darlings. However, let us not forget that these technologies are just that – technologies that are relevant in a given time and space, but not more. Yes, some flamewars are fun but in retrospect they always are 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 like FairPhone[28] may prove more important than the OS or vendor of your choice in the future.

Lost in Space

If you are lost in the jungle 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 already successful. This guide will help you make an informed decision!

28 fairphone.com

BY Anna Alfut

Conceptional Design For Mobile

Stumbling upon an idea is a wonderful Aha! moment. You suddenly know what to do and have the confidence that your idea will solve the problem faced by your potential users. These moments are usually preceded with a lot of research and experience; but it is that instant realization that probably makes people say ideas are easy. Less easy is how to transform the concept into a working product, particularly within the constraints of the medium you want to express your idea with - a mobile app. Not only do you aim to build a stable application, you also want it to be helpful and easy to use. Before you get into design and coding, it is worth spending some time in refining your idea.

Capturing The Idea

Write a **summary** that outlines your app concept in a few sentences. Try to explain it to several people (other than yourself and other team members) to see how well they can understand and relate to it.

Define your **content**. Ask yourself what the core content of your application is. Depending on the application type, it may be photos (Pinterest), user generated feeds (Twitter), text (book readers) etc. Once you recognize the main value that your app will deliver, it is easier to get the right focus in the UI. For instance, if you are creating a book reader app you probably want to make sure that the typography is of good quality.

Describe the main **functionality**. What will users do via your interface? Think about it in terms of verbs and try to list them out: browse, share, buy etc. You will notice that some activities are related. For instance, if your application has a strong community aspect there will be a number of features that you can group (like sharing, commenting, messaging, following). This can be another UI hint for you. It helps users if related functionalities are presented in a similar way.

Meet your **audience**. Who are the people that you are designing for? A useful technique is to create personas - generic profiles of your user groups that will help you understand different motivations they have in using your app.

When designing for mobile experience you need to think about the **context** in which you app will be used. And how it will affect both interface and the users. Do you think you will get users full attention, or will they be jogging at the same time? Is your app a stand alone product? Does it relate or depend on other services? What will happen if there is no internet connection? How will your app's UI handle this situation?

After answering so many questions you should have a clearer idea about the app you want to build. It is worthwhile to spend some time on research. Play with other apps that might be similar to yours. Find out how they are doing: what users think about them. This is a good way of knowing the space you are entering.

As you go further with your concept development keep reviewing your set of questions. They are a good way to keep focused and see if you are on track with what you wanted to achieve.

Designing User Experience

User Experience (UX) is how users perceive your application during and after they have used it. To design and improve the overall experience you have to think about the usage flow, functionalities, interactions and visual design. How will it all work together in your app's environment? It is thinking about the issues users might have while using your app and trying to find solutions on a prototyping stage. Design is not a linear process so iterate and apply learnings from user testing as often as you can.

Usage Flow

Some apps have very linear flow to achieve a single task (e.g. a camera app). Some might have more iterative journeys. Draw out your “ideal scenario” where user starts at a point A and, after a number of steps, ends up in a point B. You can draw flowcharts or use mock-ups to map out different flows. Try to keep the navigation as straight-forward and intuitive as possible.

Wireframes

Wireframes or Mockups are flat, sketchy versions of your interface. Their purpose is to capture functionalities and overall interface framework. They can be done with pen and paper or you can use one of the many prototyping tools that are out there.

Your wireframe for a given screen will have different states depending on a scenario. For a network error you will have different instances of the same screen. Also for a different platform your flow and screen layout will change.

You should get familiar with UI guidelines for the OS you are going to develop for. Each platform is a different environment and you should read guidelines to use the correct conventions. A classic example is the back button in the header of iOS apps vs Android back button that is always visible on the screen independently from any application user might run at the same time (sometimes it is a hard button on the device). Unless you have a strong case to do otherwise, follow best practices. Do your research and get familiar with pattern galleries that are available online. You will find platform-specific links in the respective chapters of this book.

Sketching on paper is probably the best way to start as you do not need to spend time learning new software. Drawings are easier to change and scrap. It is also a lot of fun making them. One of the advantages of dedicated wireframing applications is the ability to make your wireframes into clickable prototypes that you can test with your users.

Prototyping

A prototype is the best way to visualize and evaluate your app interactions. It does not matter whether you have a big budget or working on a personal project over the weekends. Having a fairly complete prototype version of your app is the best way to evaluate your concept or to discuss it with your customer. It is done before you spend time on developing the final code and pixel perfect designs. An agreed prototype also is a useful reference that teams can work towards without risking going too much off track.

Use whatever techniques you are the most comfortable with. There is no best way to put a prototype together. It can be a HTML build, a set of flat sketches stitched together in a clickable flow, a storyboard in PowerPoint etc.

There are plenty of tools to choose from. Some are more specialised, allowing you to draw or create clickable flows only. Some allows you to mockup, share and publish advanced prototypes. Here is a list of few applications that might help:

Application	URL	Availability
Mockflow	www.mockflow.com	free, download, online
Pencil Project	pencil.evolus.vn	free, download
Axure	www.axure.com	paid, download
Balsamiq	www.balsamiq.com	paid, download
Mockingbird	www.gomockingbird.com	paid, online
Flowella	www.developer.nokia.com/Resources/Tools_and_downloads/Other/Flowella	free, download
App in seconds	www.appinseconds.com	paid, online

Visual Design

Unless you are building a solution that uses only sound input or haptic interfaces, your app UI will rely on graphics. Applying visual design is not simply coloring wireframes, an appealing visual design will improve your app's experience and make it stand-out from amongst the masses.

Spacing and visual hierarchy improves your interface usability. Layout defines details of positioning the elements on the screen and its relation to each other. After users learn your UI it should stay consistent through the flow. For instance, if your main action button changes color from screen to screen, it forces users to re-learn conventions every every time. If that is what you want, make sure you are doing it for a good reason.

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

It is best if the company branding is interpreted in the UI in a non-obstructive way. Use the background, control's colors, maybe certain images or layout choices to add the desired look and feel. Splash screen (if present) is the place where you can display some additional graphics.

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

User Testing

The best way of validating your interface and concept is to confront it with real users as soon as possible. You do not need to wait until you have a finished and polished product. In fact you should not. It is much more difficult to accept feedback on something that you considered almost done than on a clickable prototype that you can update fairly quickly.

Ask a few people to do certain tasks using your prototype. If the app you are designing is a music player you can ask them to play a song. If you are unsure of certain functionality you can try to divert the user's attention by asking them to perform reversed tasks, like changing the selected track and picking another one instead. The main thing is to moderate, not guide. You can also run testing sessions on other apps that are currently out there. You will be surprised with how much users notice about the application that you might have never thought of.

GALACTIC
FIGHTCLUB

BY Tim Messerschmidt

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 many hardware manufacturers has made it the fastest growing smartphone operating system. According to IDC[1], 75% of all smartphones sold in Q3 2012 worldwide were based on Android. In autumn 2012 Google announced that there are 700,000 apps available in the Android Market[2] and that half a billion Android devices have been activated so far[3]. Android is also used in tablets, media players, set-top boxes, desktop phones and car entertainment systems. Some non-Android devices are also able to run Android applications with reduced functionality, such as RIM's Playbook with its Black-Berry Android runtime or the new Open Source OS Sailfish[4].

Android is an operating system, a collection of preinstalled applications and an application framework (Dalvik) 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. For instance, Android version 4.2 called "Jelly Bean" introduced a functionality called "Butter" which basically works as Triple Buffer and results in a much smoother navigation and a stable frame rate. Furthermore Google implemented

1 www.idc.com
2 www.nbcnews.com/technology/gadgetbox/google-says-there-are-700-000-android-activations-day-118326
3 mashable.com/2012/09/12/500-million-android-devices-activated/
4 www.sailfishos.org/

an improved notification system with drop-downs, floating homescreen icons which automatically get positioned if you move icons towards them and USB-Audio support. There are many more tiny improvements and additions which show that Google has succeeded in getting Android to the state where it is a stable base for the future.

One of the most discussed issues when developing for Android is fragmentation: The multitude of different devices by different manufacturers and the fast progress of the platform itself leads to uncertainty over whether your Android application will run everywhere. In addition, only a very small number of phone and tablet models support the latest OS version. However, today you will reach 96,9% of the installation base if you decide to target Android 2.2 or above[5]. To reduce fragmentation issues caused by large differences in screen size, Android 3.2 introduced a new resource descriptor called "smallestWidth" which can be used to target phones and tablets with different layout depending on their dimensions[6]. To push solid user experience and consistent appearance of Android applications, Google publishes a design guide for Android apps available at developer.android.com/design/. They also publish a more general development guide called Training[7] which helps developers to cope with the most common tasks like persisting user data.

5 developer.android.com/resources/dashboard/platform-versions.html
6 developer.android.com/guide/practices/screens_support.html#NewQualifiers
7 developer.android.com/training

Prerequisites

The main programming language for Android is Java. Beware, only a subset of the Java libraries are supported and there are many platform specific APIs. You can find answers to your "What and Why" questions online in Android's Dev Guide[8] and to your "How" questions in the reference documentation[9]. Furthermore Google introduced a section in their documentation called "Android Training"[10] which targets new developers who want to learn about best practices, where developers can learn about basics like navigation and inter-app communication or more advanced features like intelligent Bitmap downloads and optimizing for less battery drainage.

To get started, you need the Android SDK[11], which is available for Windows, Mac OS X and Linux. It contains the tools needed to build, test, debug and analyze applications. You will probably also want a good Java IDE. The Android Development Tools (ADT)[12] provide explicit support for Eclipse; other IDEs, such as IntelliJ from jetbrains.com They offer good support for development, deployment and – importantly – library projects that enable the sharing of code and resources between projects.

8 developer.android.com/guide
9 developer.android.com/reference
10 developer.android.com/training/index.html
11 developer.android.com/sdk
12 developer.android.com/tools/sdk/eclipse-adt.html

Implementation

App Architecture

An Android application can include a mix of activities, services, message receivers, and data providers; these all need to be declared in the application manifest.

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, not tied directly to a visual representation. A message receiver handles messages broadcasted by the system or other applications. A data provider is an interface to the content of an application that abstracts from the underlying storage mechanisms.

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 done by intents.

An intent bundles data, such as the user's location or a URL, with an action. These intents trigger behaviors in the platform and can be used as a messaging system in your application. For instance, the intent of showing a web page will open the browser activity. The powerful thing about this building-block philosophy is that 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 user's preference and the applications installed: Any application that declares the sharing intent as their interface can be used.

The user interface of an application is separated from the code in Android-specific xml layout files. Different layouts can be created for different screen sizes, country locales and device features without touching the Java code. To this end, localized strings and images are organized in separate resource folders. Of course you are able to define layouts in code as well.

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 such like.
— **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 or 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 and many others, including tools to analyze method trace logs, inspect layouts and test apps with random events, can be found in the tools directory of the SDK.

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

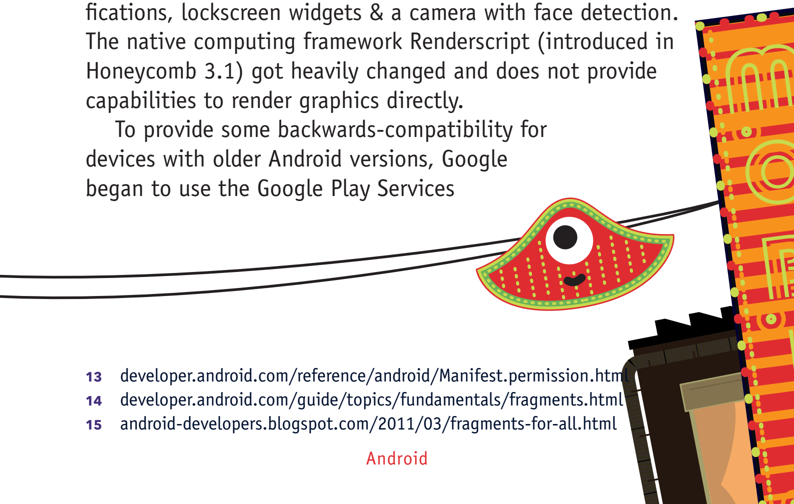
If you are facing issues, such as exceptions being thrown, be sure to check the ddms log. It enables you to check if you neglected to add all necessary permissions like `android.permission.INTERNET` in the uses-permission element[13].

If you are using features introduced after Android 2.3 – such as Fragments[14] – for large screens, be sure to add the Android Compatibility package from Google. It is available through the SDK & AVD Manager and helps to develop for Android 3.0+ without causing problems with deployment to Android 1.6[15] through to Android 2.3. Be sure to use the v4 packages in your application to provide maximum backwards support.

Developing your application against Android 3.1+, you will be able to make homescreen widgets resizable, and connect via USB to other devices, such as digital cameras, gamepads and many others.

Android Ice Cream Sandwich (4.0) and Jelly Bean (4.1 & 4.2) introduce interesting new features like expandable notifications, lockscreen widgets & a camera with face detection. The native computing framework Renderscript (introduced in Honeycomb 3.1) got heavily changed and does not provide capabilities to render graphics directly.

To provide some backwards-compatibility for devices with older Android versions, Google began to use the Google Play Services



13 developer.android.com/reference/android/Manifest.permission.html
14 developer.android.com/guide/topics/fundamentals/fragments.html
15 android-developers.blogspot.com/2011/03/fragments-for-all.html

framework[16] which gets updated via the Play Store and adds libraries like the newest Google Maps. To leverage some of the provided functionality you have to authenticate the users with OAuth 2.0.

Testing

The first step to test 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 might have changed pieces of the platform. Therefore, testing on a mix of physical devices is essential. To get an idea which devices are most popular, AppBrain's list[17] might help.

To automate testing, the Android SDK comes with some capable and useful testing instrumentation[18] 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 application after these events have occurred. MonkeyRunner[19] is a powerful and extensible test automation tool for testing the entire application. These tests can be run on virtual and physical devices.

16 developer.android.com/google/play-services/
17 www.appbrain.com/stats/top-android-phones
18 developer.android.com/guide/topics/testing/testing_android.html
19 developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

In revision 21 of SDK tools, Google finally introduced a more efficient UI automation testing framework[20] 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 currently presented views. Executing the tests is relatively easy: After you have written your test, it will be built via ANT as a JAR-file. This file has to be pushed to your device and needs to be executed via the command `adb shell uiautomator runtest`.

Open source testing frameworks, such as Robotium[21], 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[22] 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 like Jenkins or Hudson. Roboelectric runs in a standard JVM and does not need an Android run-time environment. Most other automated testing frameworks, including Robotium, are based on Android's Instrumentation framework, and will need to run in the Dalvik JVM. Plugins such as the Android Emulator Plugin[23] enable these tests to be configured and run in Hudson and Jenkins.

20 android-developers.blogspot.de/2012/11/android-sdk-tools-revision-21.html
21 code.google.com/p/robotium code.google.com/p/robotium code.google.com/p/robotium
22 pivotal.github.com/robolectric/
23 wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin

Building

Next to building your application directly in the IDE of your choice there are more comfortable ways to build Android applications. The software Gradle[24] is now the officially supported build automation tool for Android. There is also a Maven plugin[25] which gets heavily supported by the community. Both tools can use dependencies from different Maven repositories like the Maven Central Repository[26].

Signing

Your application will always be 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 application. Remember that you can use the same key for all your applications or create a new one for every app.

24 tools.android.com/tech-docs/new-build-system/using-the-new-build-system
25 code.google.com/p/maven-android-plugin/
26 www.maven.org

Distribution

After you have created the next killer application and tested it, you should place it in Android's appstore called "Play". This is a good place to reach customers and to sell your apps. Android 1.6 upwards also supports in-app purchase. This enables you to sell extra content, feature sets, et cetera, from within your app by using the existing infrastructure of Android Play[27]. It is also used by other app portals as a source for app metadata. To upload your application to Android Play, start 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 application, add screenshots and descriptions, and 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.

As there are lots of competing applications in Android Play, you might want to use alternative application stores[28]. They provide different payment methods and may target specific consumer groups. One of those markets is the Amazon Appstore, which comes preinstalled on the Kindle Fire tablet family.

27 developer.android.com/guide/google/play/billing/
28 wipconnector.com/index.php/appstores/tag/android

Monetization

Next to selling an app in one of the many available application stores there are lots of different ways of monetizing an Android application. 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 like Google's own service[29] which utilizes the Google Play Store or PayPal's Mobile Payments Library[30]. Most services differ in transaction-based fees and the possibilities they offer e.g. subscriptions, parallel payments or pre-approved payments.

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

29 developer.android.com/google/play/billing/
30 www.x.com/developers/paypal/products/mobile-payment-libraries

AWESOME

BY Ovidiu Iliescu & Michael Koch

BlackBerry Java Apps

The Ecosystem

The BlackBerry platform is developed by Canadian company Research In Motion (RIM)[1] and was launched in 1999. BlackBerry devices became extremely popular because they were equipped with a full keyboard for comfortable text input (which spawned a condition named BlackBerry Thumb[2]), offered a robust push service for email and other data, offered long battery life and included BlackBerry Messenger, their mobile social network offering. Add PDA applications such as address book, secure email, calendar, tasks and memopad to these features and you will understand why the platform was very popular among business and mainstream users alike.

The overall market share of BlackBerry phones has continued to decline in 2012[3]. To gain back lost ground in the market, RIM decided to take radical measures and introduce a completely new operating system: BlackBerry 10. The first BB10 devices have been shipped in Q1 2013. This chapter concentrates on developing apps for the older, non-BB10 BlackBerry devices on the market. BB10 development is explained in a separate chapter.

BlackBerry OS is the operating system found on all BlackBerry smartphones released before 2013. Its latest iteration released in early 2012 (BlackBerry OS 7.1) offers some notable improvements over its predecessor: tethering, WiFi calling support, NFC Tag support and FM radio.

1 rim.com
2 wikipedia.org/wiki/Blackberry_thumb
3 gs.statcounter.com

The most relevant API additions in OS 7.1 are:

- NFC Peer-to-Peer API, which offers the ability to initiate data transfers between two devices via NFC, then complete the transfer via Bluetooth
- FM Radio API
- Profiles API, which allows read/write access to the user's current profile

For BlackBerry OS, two development approaches are available depending on the type and nature of your planned project. For mid-sized to large applications native Java development is the best choice; while small apps could be developed with the BlackBerry WebWorks SDK.

Although it will be phased out in the future, currently the BlackBerry Java API is the most commonly used method to develop BlackBerry apps. As such, this chapter focuses on Java development.

Prerequisites

As for all Java-driven application development, you need the Java SDK[4] (not the Java Runtime Environment). Next, you need Eclipse and the BlackBerry plugin[5]. You can download these separately or you can download the convenient bundle that RIM provides, which includes both. The bundle also includes the SDK and simulators for the latest BlackBerry OS, with instructions on how to download older SDKs (needed for targeting older devices) available on the download page. Additionally,

4 oracle.com/technetwork/java

5 us.blackberry.com/developers/javaappdev/javaplugin.jsp

extra device simulators are available for download from RIM's website[6].

To deploy your app package on to a device for testing you should download and install the BlackBerry Desktop Manager[7]. For faster deployment, you might also use a tool called javaloader that comes with the JDE.

Implementation

The BlackBerry JDE is partly based on Java ME and some of its JSR extensions: Integrated into the SDK is the MIDP 2.0 standard with popular JSR extensions that provide APIs for UI, audio, video, and location services among others[8]. This means that BlackBerry apps can be created using Java ME technologies alone.

Another option is to use BlackBerry's proprietary extensions and UI framework that enable you to make full use of the platform.

Native UI components can be styled to an extent, but they inherit their look from the current theme. This can be prevented in code, by overriding the `Field.applyTheme()` method for each component/field.

From OpenGL-ES to homescreen interaction and cryptography, the BlackBerry APIs provide you with everything you need to create compelling apps. In addition to the official BlackBerry tools, there are third party extensions that enable you to enhance your apps, for example J2ME Polish[9] or Glaze[10] which enable you to design and animate your UI using CSS.

6 us.blackberry.com/sites/developers/resources/simulators.html
7 us.blackberry.com/apps-software/desktop/
8 blackberry.com/developers/docs/6.0.0api/index.html
9 j2mepolish.org
10 glaze-ui.org

Services

BlackBerry offers many services that can be useful in developing your applications including advertising, mapping, payment and push services[11].

The push service[12] is useful mainly in mail, messaging or news applications. Its main benefit is that the device waits for the server to push updates to it, instead of the device continuously polling the server to find out if updates are available and then pulling the updates from the server. This reduces network traffic, battery usage and, for users on metered data plans or roaming, lowers costs. It works as follows: Your server sends a data package of up to 8KB to the BlackBerry push infrastructure. The infrastructure then broadcasts the message to all or a group of clients (for content such as a news report) or to one specific client (for content such as a chat message). The device client then receives the message through BlackBerry's Push API and may confirm message receipt back to the infrastructure. Your server can then check if the message was delivered. BlackBerry offers the push mechanism as a limited free service, with a premium paid extension that enables you to send more push messages.

Porting

Porting apps between BlackBerry devices is easy because the OS is made by a single company that has been careful to minimize fragmentation issues. However, this does not entirely eliminate challenges:

— Some classes and functionality are only available on specific OS versions. For example the FilePicker that is used to choose a file is only available from OS 5.0 onwards.

11 developer.blackberry.com/services/#platform
12 us.blackberry.com/developers/platform/pushapi.jsp

- You need to handle different screen resolutions and orientation modes (landscape and portrait).
- You need to handle touch and non-touch devices. In addition, the Storm devices use a touchscreen that is physically clickable, so there is a distinction between a touch and a click on these devices. BlackBerry's more recent touch devices do not use this technology anymore.

Porting to other Java platforms such as Java ME and Android is complicated as it is not possible to port the BlackBerry UI.

Code written for server communication or storage might be reused on Java ME and Android if you avoid native BlackBerry API calls. In general, cross-platform portability strongly depends on how frequently your app uses native BlackBerry components. For example it is not possible to reuse BlackBerry push services classes on other platforms.

Testing

BlackBerry provides simulators for various handsets either bundled with the Eclipse plugin or as separate downloads. These simulators enable you to run an app on a PC in the same way it would be run on a device. The Blackberry testing and debugging capabilities are on par with those of other modern platforms such as Android and iOS: the simulators allow developers to simulate a large variety of events (incoming calls, changes to GPS coordinates, changes to network conditions, etc) while on-device debugging makes it easy to test your code on real hardware.

In addition, automated testing is also possible, though somewhat limited and complicated. You can use the bundled

FledgeController tool[13] to inject events programatically from your computer, or you can use the EventInjector class[14] to inject events from a BlackBerry application running on the device (or simulator). However, there is very little documentation available on the topic, so expect some hacking and head-scratching to be a part of your BlackBerry automated testing experience.

Signing

Many security-critical classes and features of the platform (such as networking or file APIs) require an application to be signed such that the publisher can be identified. To achieve this, you need to obtain a signing key directly from BlackBerry[15]. The signing itself is undertaken using the rapc tool, which also packages the application.

Distribution

BlackBerry's own distribution channel is called App World[16] where you can publish your apps. For paid applications, you'll get a 70% revenue share. In addition GetJar[17] is a well-known independent website that also publishes BlackBerry apps.

13 docs.blackberry.com/en/developers/deliverables/15476/Using_the_BBSmrtphnSmltr_programmatically_607582_11.jsp
14 blackberry.com/developers/docs/4.1api/net/rim/device/api/system/EventInjector.html
15 blackberry.com/SignedKeys/
16 appworld.blackberry.com
17 getjar.com

Learn More

If you want to learn more about BlackBerry Java development, the following are a few resources that might help you.

Bundled sample apps

The SDKs come with a great selection of sample apps, showcasing everything from simple "Hello, World!" applications to a complex geo-location and multimedia apps.

Online

A number of important online resources are available:

- The official BlackBerry documentation microsite[18]
- The BlackBerry developer forums[19]

There is also a wealth of BlackBerry knowledge scattered around the internet, dealing with some problems and topics much better than the official documentation. Search engines are your friends!

Books

Printed works dealing with BlackBerry Java development include:

- **BlackBerry Development Fundamentals**[20] by John Wargo
- **Beginning BlackBerry 7 Development 2nd Edition** by Anthony Rizk
- **Advanced BlackBerry 6 Development 2nd Edition** by Chris King

18 developer.blackberry.com/java/documentation/
19 supportforums.blackberry.com/t5/Java-Development/bd-p/java_dev
20 bbdevfundamentals.com/

Pancake

BY Marcus Ross

BlackBerry 10

The Ecosystem

The BlackBerry 10 platform (BB10) is a general relaunch from RIM. BB10 devices came to market in Q1 2013 - there are no upgrade plans for older generation devices. RIM has taken this approach in order to catch-up with competing mobile operating systems: iOS, Android and Windows Phone 8. Another major goal is to harmonize RIM's mobile phones and tablets by having them run the same operating system and the same apps.

RIM is under massive market pressure and is investing a lot in this relaunch. They have to make it a success if they do not want to lose even more ground in the mobile market. This means new and interesting opportunities for app developers who are willing to develop for the new platform. Although the OS is entirely new, in its core it is based on QNX, a realtime os for embedded devices. The other parts of the BlackBerry ecosystem, like the App World or the push-service, have not changed. Security-wise, BB10 is built to the same high standards for which the BlackBerry platform is well known.

Development

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

- C Native SDK
- C++ Cascades SDK
- HTML5 (WebWorks SDK)
- Adobe Air
- Android Runtime (Android 2.3.3 compatibility layer)
- Blackberry App Generator

To attract developers to their new OS, RIM provides a rich set of resources including a simulator, many sample projects on GitHub[1] and frequently updated documentation[2].

A major point of discontent, for which RIM has received a lot of backlash, is that the current Java API is no longer supported. This means that Java developers writing code for current BlackBerry devices need to re-orient themselves to one of the technologies previously mentioned. As not all developers will be willing to do this, there is concern in the community that too many developers will "jump ship" and re-orient themselves to competing platforms. Furthermore, since there is no migration path for current generation apps, developers will need to rewrite them from scratch for the new platform. This is necessary because the core of the new OS is based on QNX[3], a realtime embedded OS. On the other hand, the new platform offers new opportunities, e.g. for web developers and Android developers who can easily migrate their apps.

1 github.com/blackberry
2 developer.blackberry.com/platforms/bb10
3 www.qnx.com

C Native SDK

The BlackBerry NDK supports many open standards that allow developers to bring their existing apps to the platform. To get started, there is a Native Dev Site[4]. Writing your code with Native SDK enables your app is as close to the hardware as possible. The BlackBerry 10 Native SDK includes everything you need to develop programs that run under the BlackBerry 10 OS: a compiler, linker, libraries, and an extensive Integrated Development Environment (IDE). It is available for Windows, Mac and Linux.

The core development steps are the following:

- Request a signing account and keys
- Set up the native SDK[5]
- Install and configure the simulator[6]
- Configure your environment for development and deployment
- Create your first project
- Run sample applications

4 developer.blackberry.com/native/beta
5 developer.blackberry.com/native/download
6 developer.blackberry.com/native/download

As a new addition, Blackberry added Scoreloop[7] support to the NDK. Scoreloop is a technology that enables mobile social gaming. It lets developers integrate social features into their games, while preserving each game's specific look and feel. Some of the features currently available include:

- User profile
- Leaderboards
- Challenges
- Awards and achievements

C++ Cascades SDK

Developing with C++ and Cascades is another option. Cascades has been designed to allow developers to build a BlackBerry native application with strong support for easy UI implemenation. The Cascades framework separates application logic from the UI rendering engine. In an application, the declared UI controls, their properties and their behavior are defined in an Markup-Language called QML[8]. When your application runs, the UI rendering engine displays your UI controls and applies any transitions and effects that are specified. The Cascades SDK provides the following features:

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

7 developer.blackberry.com/native/documentation/bb10/com.qnx.doc.scoreloop.lib_ref/topic/overview.htmll

8 en.wikipedia.org/wiki/QML

- Storage and retrieval of data
- Certificate managing and cryptographic tools

The Cascades framework is built using the Qt application framework. This architecture allows Cascades to leverage the Qt object model, event model, and threading model. The slots and signals mechanism in Qt allows for powerful and flexible inter-object communication. The Cascades framework incorporates features of fundamental Qt classes (such as QtCore, QtNetwork, QtXml, and QtSql, and others) and builds on them. This lets developers define things instead of programming them e.g. they only need to define the duration and type of an animation, instead of programming it. This approach is similar to iOS with Core Animation. Even QML can be written by experienced Javascript developers because of their JSON-like markup.

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

If you are a designer, the Cascades Exporter[9] is for you. This Adobe Photoshop plugin slice and rescale your images and package them up to a .tmz file (compressed, sliced and metadata enhanced image assets). These asset files can be easily used by a developer with the QNX Momentics IDE.

To get further information, there is a Cascades Dev Site[10] available.

9 developer.blackberry.com/cascades/documentation/design/cascades_exporter
10 developer.blackberry.com/cascades

HTML5 WebWorks

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

The first tool is the WebWorks SDK[11]. Among other features, it allows you to write regular webpages and then package them as native BlackBerry apps with ease. If you want to mimic the Blackberry-UI style in HTML, there is a project on GitHub to help you. It is called BBUi.js[12]. It provides extensive CSS to make your regular webpage look like a native Blackberry-UI application. You use data-attributes to enhance the HTML for that approach.

The second tool is the Ripple Emulator[13]. It is a Chrome Browser extension that acts as a Blackberry 10 device simulator for WebWorks apps. It also emulates hardware-specific features, such as the accelerometer and the GPS sensor. You can even use it to package and deploy your app without going through the command-line.

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

To get more information about developing with WebWorks there is a HTML5 Dev micro-site[14] with more information.

Adobe Air

If you are an existing AIR develeoper you can add BB10 as a new distribution channel. You will use the BlackBerry 10 SDK for Adobe AIR to create applications for BlackBerry devices.

11 developer.blackberry.com/html5/download/sdk
12 github.com/blackberry/bbUI.js
13 developer.blackberry.com/html5/download/ripple
14 developer.blackberry.com/html5

You can use the SDK with Adobe ActionScript and Adobe Flex APIs to create/port Blackberry Apps. These APIs provide some unique UI components and predefined skins, as well as listeners for events that are specific to BlackBerry devices. Using the Adobe Flash Builder APIs, your application can also access the features that are unique to mobile devices, such as the accelerometer and geolocation information. Additionally, you can harness the features of the BlackBerry 10 Native SDK by developing AIR Native Extensions (ANE).

To begin developing your Adobe AIR application:

- Download and install VMware Player for Windows or VMware Fusion for Mac
- Download the BlackBerry 10 Simulator
- Download the BlackBerry 10 SDK for Adobe AIR
- Begin development with Adobe Flash Builder, Powerflasher FDT or Command Line Tools

For further information, visit the dedicated website[15].

Android Runtime

You can use the BlackBerry Runtime for Android apps to run Android 2.3.3 platform applications on BlackBerry 10. To use the runtime, you must first repackage your Android applications in the BAR file format, which is the file format required for an application to run on BlackBerry 10.

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

15 developer.blackberry.com/air/

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

If you want to find out more about running Android apps on BB10, please visit the dedicated website[16].

16 developer.blackberry.com/android

Blackberry App Generator

If you are not a developer, RIM provides an easy way of generating a simple app for BB10 with the Blackberry App Generator[17]. This webpage generates an app based on imput-sources like:

- RSS feeds
- Tumbler
- Facebook
- YouTube
- flickr
- and more

It generates a master-detail styled app that can be customized with a logo and color selection. For a simple news-app that approach is totally fine, but do not expect any "CNN"-like masterpieces.

17 blackberryappgenerator.com/blackberry/

Testing

BlackBerry continues to provide a simulator for BB10 handsets as a separate downloads[18]. This simulator enables you to run an app on a PC/Mac/Linux in the same way it would be run on a real BlackBerry device. To assist with testing, the simulator comes with a little application called controller. This utility enables you to simulate things like setting the battery level, GPS-position, NFC or tilting the device and thereby check how your application reacts in real-world scenarios.

Signing

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

If you like to test your unsigned app on a physical device, you need to request a file called debug token. This token enables a specific BB10 device to run unsigned apps. For this setup procedure you need to request a signing file (client-PBDT-xxxxx.csj) via Blackberry Key Order Form[19]. After receiving the file by email you can install a debug token with the command-line tools. After this setup you can run unsigned apps on your device. Please be advised that this needs to be done on each device separately.

If you would like to publish your app in BlackBerry's AppWorld, you need a signing key. The signing keys can be

18 developer.blackberry.com/devzone/develop/simulator/
19 www.blackberry.com/SignedKeys/codesigning.html

ordered through the Blackberry Key Order Form[20]. To help you with this process of setup Blackberry provides a step by step webpage[21] that guides you through the process.

Distribution

As with all previous OS versions, BB10 apps are distributed via Blackberry App World[22]. The necessary vendor account can be created at the Vendor Portal for BlackBerry App World[23].

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

The second option is an enterprise distribution. This lets you roll out an internal app in your organization instead of making it publicly available to any user. This is suitable for B2B Apps. If you want to find out more about enterprise distribution, please visit the dedicated website[24].

20 www.blackberry.com/SignedKeys/codesigning.html
21 developer.blackberry.com/CodeSigningHelp/codesignhelp.html
22 appworld.blackberry.com
23 appworld.blackberry.com/isvportal
24 developer.blackberry.com/distribute/enterprise_application_distribution.html

MOBILE

BY Alexander Repty

iOS

The Ecosystem

A Small History of iOS

Apple announced iOS (which was then simply referred to as OS X, after Mac OS X, the operating system for Apple's Macintosh computer line) at MacWorld 2007 alongside the first iPhone, which was released, with iPhone OS 1.0, on June 29, 2007. Ever since, Apple has released a new generation of the iPhone accompanied by a new major release of iOS every year, at some point between June and October. In September 2012, the latest major release (6.0) was released.

Devices Running iOS

Currently, Apple sells several distinct devices (in various configurations) that run iOS:

- iPhone
- iPod touch
- iPad
- iPad mini
- Apple TV

With the exception of Apple TV, all of those devices include the App Store and can run 3rd party applications.

Most devices will run the most current version of iOS for two years or more after their initial release, so developers should consider this when planning to develop an application. Using older hardware generally means fewer resources, such as CPU cycles and RAM; and in some cases different display sizes and/or screen resolutions.

A detailed list of iOS devices, their capabilities and supported iOS versions can be found on Wikipedia[1].

Device & App Sales

According to information from Apple, which are usually milestones announced at special media events, they have sold over 400 million iOS devices through June 2012. Since sales of iOS devices are still gaining momentum, even after five years, a large number of those devices can be assumed to be in active use and still running either iOS 5 or iOS 6.

As of January 2013, the App Store contains over 775,000 applications by 3rd party developers, which have been collectively downloaded over 40 billion times, paying out over seven billion dollars to developers according to Apple[2].

Technology Overview

Frameworks & Language(s)

Since iOS builds on the foundation of Mac OS X, it uses a lot of the same frameworks and technologies, except for the Cocoa Touch layer (which manages and draws the user interface) and a number of other, small frameworks that are unique to either of the systems. This makes it easy for a number of applications to use a similar code base and just vary on the user interface, which would have to be completely redesigned for touch devices anyway.

Most Apple-supplied frameworks for iOS are written in Objective-C (or supply Objective-C APIs over a different back-

1 en.wikipedia.org/wiki/List_of_iOS_devices

2 www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html

end), which is a Smalltalk-inspired lightweight runtime on top of C and retains full C compatibility. Few frameworks supply C APIs, mostly those used for audio and video programming. The system also supports development in C++ and Objective-C++ and includes standard frameworks for all of those languages.

Before the release of iOS, Objective-C lead a somewhat shadowy existence with ratings as low as 0.03% in the TIOBE index[3], thanks to its usage almost exclusively for Apple's desktop platform, Mac OS X. In December 2007, it was only the 57th most popular programming language and since has made its way to number three in 2012, just behind Java and C after winning "Programming Language of the Year" in 2011. Its popularity continues to rise way more quickly than Java and C.

Over the past years, Apple has made numerous improvements both to the Objective-C runtime and the LLVM compiler to add new features to the language, such as automatic memory management, blocks (a form of closures) and automatically-synthesized properties, most of which are improvements developers directly benefit from by having to write less code.

On their developer website[4], Apple provides a plethora of resources for iOS developers, including software downloads, training videos, getting-started guides, documentation, sample code and forums.

Most of these resources contain very valuable information, such as the Human Interface Guidelines, which every developer should have read.

3 www.tiobe.com/index.php/content/paperinfo/tpci/index.html
4 developer.apple.com/devcenter/ios/

Xcode and Alternatives

For iOS (and Mac OS X) development, Apple supplies its own suite of developer tools, completely free of charge, including the following applications:

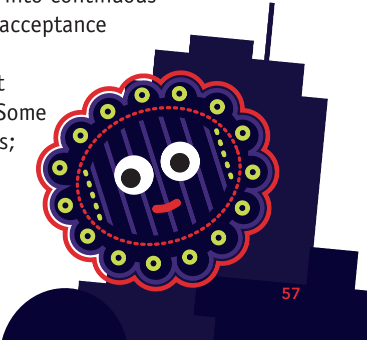
- **Xcode:** integrated development environment
- **Instruments:** performance analyzer running on top of DTrace
- **Dashcode:** development environment for Dashboard widgets (Mac OS X) and other HTML-related content
- **iOS Simulator:** simulates an iOS environment for quick testing

A commercial alternative IDE to Xcode is JetBrains' AppCode[5], a Java application with various more in-depth features than Xcode has to offer. For those who are looking to avoid Objective-C entirely, there are complete environments such as MonoTouch[6], which even offer cross-platform support.

Testing & Debugging

The iOS developer tools include support for unit testing as well as automated user interface testing via the UIAutomation framework. Through Xcode's command line tools, those tools can even be integrated into continuous integration systems for automated acceptance testing.

There are numerous external test automation tools and frameworks. Some are proprietary commercial offerings; however the majority are now



5 www.jetbrains.com/objc/
6 xamarin.com/monotouch

opensource even from commercial companies who hope to sell services to make your automated testing easier and more powerful. Many of the external test automation tools require the developer to incorporate a library into a special build of their app. The library allows the tests to interact with your app. Be careful to keep the special builds separate from builds intended for release to the app store, otherwise you may have an unwelcome rejection when trying to submit your app to the app store.

Xcode includes both gdb and lldb and will automatically use the appropriate one based on which compiler is being used for the application. Although the developer retains complete control over the debugger using the prompt, Xcode offers some user interface for often-used actions, such as setting, editing and deleting breakpoints and viewing variables and memory contents.

Instruments also contains various features to help developers hunt down bugs, performance bottlenecks, and memory management problems.

As the name implies, the iOS Simulator is just that - a simulator and as such it has different runtime characteristics than actual iOS devices, and the simulator does not emulate a complete iOS environment. Thus, a number of issues that will appear on real-world devices simply won't surface when testing applications in the simulator. Fortunately, all supported tests can be executed on actual devices, too, and Apple allows developers to provision up to 100 iOS devices to run their applications on for testing and demonstration purposes.

Recruiting and managing third-party software testing (if not available locally in-house) are made easy through TestFlight[7] and HockeyApp[8], both of which offer various helpful features,

7 testflightapp.com/
8 hockeyapp.net

such as automatic code signing, crash report collection and in-app updating for beta testers.

Learn More

Blogs & Newsletters

A bunch of developers regularly post valuable information about current developments on their blogs. One notable blog is Mike Ash's[9], on which he posts a very interesting series of Q&As about Objective-C and Cocoa development. A number of other great other blogs can be found through *www.planetcocoa.org*.

iOS Dev Weekly[10] is one notable newsletter that compiles the most interesting news, open source code, tools as well as design and marketing advice in a handy list and sends it out to developers every Friday.

Conferences

Owing to the growing popularity of iOS, there are numerous iOS-centered conferences around the world every year, way too many to list here. There are two notable conferences though that deserve a mention:

— Every year in June, Apple holds their Worldwide Developer Conference (WWDC)[11]. The full-week conference in San Francisco includes many simultaneous tracks about Mac OS X and iOS development with sessions by Apple engineers as well as on-hands labs, where attendees can ask Apple

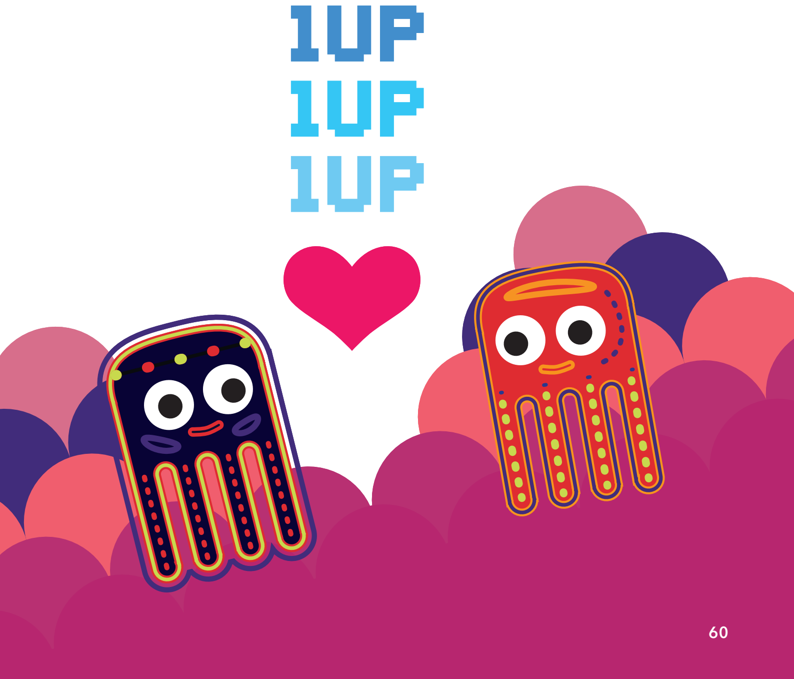
9 www.mikeash.com/pyblog/
10 iosdevweekly.com
11 developer.apple.com/wwdc/about/

engineers for advice about problems they're facing while developing their apps.

- The biggest and most successful European conference around Mac OS X and iOS development is the NSConference[12], held every year around March in England.

Both of these conferences usually sell out in a matter of days, if not hours - so plan well ahead and subscribe to alerts about the tickets going on sale if you are planning to go to either of those conferences.

12 ideveloper.tv/nsconference/



café
Colossal

BY Ovidiu Iliescu

Java ME (J2ME)

The Ecosystem

J2ME (or Java ME as it is officially called) is the oldest mobile application platform still widely used. Developed by Sun Microsystems, which has since been bought by Oracle, J2ME is designed to run primarily on feature phones. It has been very successful in this market segment, with an overwhelming majority of feature phones supporting it. J2ME is also supported natively on Symbian and current BlackBerry smartphones not based on BB10.

J2ME's major drawback is that, due to its age and primary market segment, it does not fare well compared to modern smartphone platforms, such as Android, iPhone, BlackBerry and Windows Phone: it offers a less powerful set of APIs, often runs on less powerful hardware and tends to generate less money for the developer. As a consequence, J2ME's popularity in the developer community has declined significantly in recent years.

So why would you want to develop for J2ME? Mainly for one reason: market reach. In Q2 2012, smartphone sales still accounted only for 36.7% of total mobile phone sales worldwide[1]. The majority of devices are still feature phones which usually support Java ME. So if your business model relies on access to as many potential customers as possible, then J2ME might still be a great choice - especially when you are targeting markets like certain African countries or India.

However, if your business model relies on direct application

1 gartner.com/it/page.jsp?id=2120015

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

Prerequisites

To develop a Java ME application, you will need:

- The Java SDK[2] (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers[3], NetBeans[4] with its Java ME plug-in or IntelliJ[5]. Beginners often chose NetBeans.
- An emulator, such as the Wireless Toolkit[6], the Micro Emulator[7] or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard[8]. If you build applications professionally you will probably want to use a build tool such as Maven[9] or Ant[10] also.
- You may want to check out J2ME Polish[11], the open source framework for building your application for various devices.

2 oracle.com/technetwork/java/javame/downloads
3 eclipse.org
4 netbeans.org
5 jetbrains.com
6 oracle.com/technetwork/java/download-135801.html
7 microemu.org
8 proguard.sourceforge.net
9 maven.apache.org
10 ant.apache.org
11 j2mepolish.org

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

Also download and read the JavaDocs for the most important technologies and APIs: You can download most Java-Docs from *www.jcp.org*. For manufacturer-specific APIs, documentation is usually available on the vendor's website (for example, the Nokia UI API[12]).

Implementation

The Java ME platform is fairly straight-forward: it comprises the Connected Limited Device Configuration (CLDC)[13] and the Mobile Internet Device Profile (MIDP)[14], both are quite easy to understand. These form the basis of any J2ME environment and provide a standardized set of capabilities to all J2ME devices. As both CLDC and MIDP were designed a decade ago, the default set of capabilities they provide is rudimentary by today's standards.

Manufacturers can supplement these rudimentary capabilities by implementing various optional Java Specification Requests (JSRs). JSRs exist for everything from accessing the device's built in calendar, address book and file system (JSR 75); to using the GPS (JSR 179) and Near Field Communication (JSR 257). For a comprehensive list of JSRs related to Java ME development, visit the Java Community Process' List by JCP Technology[15].

It is very important to know that the JSRs you want to use may not be available for all devices; and implementations vary

12 www.developer.nokia.com/Community/Wiki/Nokia_UI_API
13 java.sun.com/products/cldc/overview.html
14 java.sun.com/products/midp/overview.html
15 jcp.org/en/jsr/tech?listBy=1&listByType=platform

significantly from one phone model to another; so capabilities available on one device might not be available on another device, even if the two devices have similar hardware.

The Runtime Environment

J2ME applications are called MIDlets. A MIDlet's lifecycle is quite simple: it can only be started, paused and destroyed. On most devices, a MIDlet is automatically paused when minimized; it cannot run in the background. Some devices support concurrent application execution, so it is possible for applications to run in the background. However, this usually requires the use of vendor-specific APIs and/or relies on device-specific behavior, which can cause fragmentation issues.

MIDlets also run in isolation from one another and are very limited in their interaction with the underlying operating system – these capabilities are provided strictly through optional JSRs (for example, JSR 75) and vendor-specific APIs.

Creating UIs

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

1. Highlevel LCDUI components: you use standard UI components, such as Form and List
2. Lowlevel LCDUI: you manually control every pixel of your UI using low-level graphics functions
3. SVG: you draw the UI in scalable vector graphics then use the APIs of JSR 226[16] or JSR 287[17].

In addition, you will find that some manufacturers provide additional UI features. For example, Nokia's latest feature

16 www.jcp.org/en/jsr/detail?id=226
17 jcp.org/en/jsr/detail?id=287

phone series (Nokia Asha) employ either the Full Touch[18] or the Touch and Type[19] user interface paradigms, depending on device model. The Nokia UI API was extended in order to enable developers to make best use of these UI paradigms in their applications. Similarly Samsung provide pinch zoom features in their latest Java ME APIs[20].

There are also tools that can help you with the UI development. All of them use low-level graphics to create better looking and more powerful UIs than are possible with the standard highlevel LCDUI components.

1. **J2ME Polish[21]:** This tool separates the design in CSS and you can use HTML for the user interface. It is backward-compatible with the highlevel LCDUI framework
2. **LWUIT[22]:** A Swing inspired UI framework
3. **Mewt[23]:** Uses XML to define the UI

One very important aspect to consider when designing your UI is the typical screen resolution for Java ME devices. The vast majority of Java ME devices have one of the following resolutions: 240x320, 176x208, 176x220, 128x160, 128x128 or 360x640 pixels. By far the most popular is 240x320, while 360x640 is a common resolution for high-end Java ME devices (typically those running Symbian or Blackberry) and 176x208/220 is a common resolution for low-end devices. You will also encounter devices that have these resolutions in landscape, for example 320x240 instead of 240x320 pixels.

Handling so many different resolutions can be a challenge.

18 www.developer.nokia.com/Resources/Library/Full_Touch/
19 www.developer.nokia.com/Community/Wiki/Nokia_UI_API_1.1b
20 developer.samsung.com/java/technical-docs/Multi-Touch-in-Samsung-Devices
21 j2mepolish.org
22 lwuit.java.net/
23 mewt.sourceforge.net

Your best approach is to create UI layouts that can scale well across all of them, in the same way that web pages scale well across different browser window sizes. You can also create custom UIs for each resolution, though this is not recommended because it is time consuming, error prone and expensive.

Another aspect worth considering is the size of your application's assets, especially its graphical assets. Whenever possible, your assets should be optimized, in order to keep your application's size as small as possible. This results in cheaper downloads for your users (as less data traffic is needed) and greater market reach (as some devices have a limit on the maximum application size). A great free tool for this is PNG-Gauntlet[24], which can optimize your graphical assets without compromising quality.

Despite the platform's limitations, it is quite possible to create great looking and easy to use Java ME user interfaces, particularly if one of the tools mentioned above is used.

Testing

Because of the fragmentation in the various implementations of Java ME, testing applications is vital. Test as early and as often as you can on a mix of devices. Some emulators are quite good (personal favorites are BlackBerry and Symbian), but there are some things that have to be tested on devices.

Thankfully, vendors like Nokia[25] and Samsung[26] provide subsidized or even free remote access to selected devices.

24 pnggauntlet.com
25 forum.nokia.com/rda
26 innovator.samsungmobile.com

Automated Testing
There are various unit testing frameworks available for Java ME, including J2MEUnit[27], MoMEUnit[28] and CLDC Unit[29]; System and UI testing is more complex given the security model of J2ME, however JInjector[30] is a flexible byte-code injection framework that supports system and UI testing. Code coverage can also be gathered with JInjector.

Porting

One of the strengths of the Java environment for mobile devices is that it is backed by a standard, so it can be implemented by competing vendors. The downside is that the standard has to be interpreted, and this interpretation process can cause differences in individual implementations. This results in all kinds of bugs and non-standard behavior. In the following sections we outline different strategies for porting your applications to all Java ME handsets and platforms.

Lowest Common Denominator
You can prevent many porting issues by limiting the functionality of your application to the lowest common denominator. In the J2ME world this usually means CLDC 1.0 and MIDP 1.0. If you only plan to release your application in more developed countries / regions, you may consider targeting CLDC 1.1 and MIDP 2.0 as the lowest common denominator (without any additional APIs or JSR support).

Depending on the target region for the application you might also consider using Java Technology for the Wireless

27 j2meunit.sourceforge.net
28 momeunit.sourceforge.net
29 snapshot.pyx4me.com/pyx4me-cldcunit
30 code.google.com/p/jinjector

Industry (JTWI, JSR 185) or the Mobile Service Architecture (MSA, JSR 248) as your baseline. Both extensions are designed to ensure a common implementation of the most popular JSRs. They are supported by many modern devices and provide many more capabilities to your applications. However, in some regions such as Africa, South America or India you should be aware that using these standards may limit the number of your potential users, because the more common handsets in these regions do not implement those extensions.

Using the lowest common denominator approach is typically easy: There is less functionality to consider. However, the user experience may suffer if your application is limited in this way, especially if you want to port your application to smartphone platforms later. So this approach is a good choice for simple applications – for comprehensive, feature-rich applications it is not the way to go.

Porting Frameworks

Porting frameworks help you deal with fragmentation by automatically adapting your application to different devices and platforms. Such frameworks typically feature the following components:

- Client libraries that simplify development
- Build tool chains that convert code and resources to application bundles
- Device databases that provide information about devices
- Cross compilers to port your application to different platforms

For Java ME some of the options you can choose from are: Celsius from Mobile Distillery[31] that is licensed per month,

31 mobile-distillery.com

Bedrock from Metismo[32] that provides a suite of cross compilers on a yearly license fee and J2ME Polish from Enough Software[33] that is available under both the GPL Open Source license and a commercial license. Going in the other direction (from C++ to Java ME) is also possible with the open source MoSync SDK[34].

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

Good frameworks enable you to use platform and device specific code in your projects, so that you can provide the best user experience. In other words: a good porting framework does not hide device fragmentation, but makes the fragmentation more manageable.

Signing

The Java standard for mobile devices differentiates between signed and unsigned applications. Some handset functionality is available to trusted applications only. Which features are affected and what happens if the application is not signed but uses one of those features, is largely dependent on the implementation. On one phone the user might be asked once to enable the functionality, on another they will be asked every time the feature is used and on a third device they will not be able to use the feature at all without signing. Most implementations also differentiate between the certification authorities who have signed an application.

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

32 metismo.com
33 enough.de
34 mosync.com

certificate are similarly trusted on devices with that carrier's public key certificate installed.

Applications signed by JavaVerified[35], Verisign[36] or Thawte[37] are on the lowest security level.

To make matters worse, not every phone carries all the necessary root certificates. And, in the past, some well known device vendors have even stripped away all root certificates. The result is something of a mess, so consider signing your application only when required, that is when deploying to an app store or when you absolutely need access to security constrained features. However, in some cases an app store may offer to undertake the signing for you, as Nokia Store does.

Another option is to consider using a testing and certification service provider and leaving the complexity to them. Intertek[38] is probably the largest such supplier.

Distribution

J2ME applications can be installed directly onto a phone in a variety of ways; the most commonly used methods are over a Bluetooth connection, via a direct cable connection or Over-the-Air (OTA). However, app stores are probably the most efficient way to distribute your apps. They manage the payment, hosting and advertisements, taking a revenue share for those services. Some of the most effective stores include:

35 javaverified.com
36 verisign.com
37 thawte.com
38 intertek.com/wireless-mobile

- Handmark[39] and Mobile Rated[40] provide carrier and vendor independent application stores.
- GetJar[41] is one of the oldest distributors for free mobile applications - not only Java applications.
- Nokia Store[42] targets Nokia users worldwide and provides a revenue share to the developer at 70% from credit card billing and 60% from operator billing
- Carriers are in the game also, such as Orange[43] and O2[44].

Basically almost everyone in the mobile arena has announced an app store. An overview of the available app stores (not those selling J2ME apps alone) can be found in the WIP App Store Catalogue[45]. Also see the separate chapter on Appstores in this guide to learn more.

Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia[46] and Futurlink[47].

Learn More

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

39 store.handmark.com
40 mobilerated.com
41 getjar.com
42 publish.ovi.com
43 www.orangepartner.com/distribute
44 mobileapps.o2online.de
45 wipconnector.com/appstores/
46 waymedia.it
47 www.futurlink.com

Online

As Java ME is one of the oldest mobile platforms still used, it's easy to find tutorials and resources related to it, for example those available from J2ME Salsa[48].

In addition, there is a rich open source scene in the J2ME sector. Interesting projects can be found via the blog at *opensource.ngphone.com*.

You will also find fascinating projects on the Mobile and Embedded page of java.net[49], for example the Bluetooth project Marge[50].

Books

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

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

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

48 j2mesalsa.com

49 community.java.net/mobileandembedded/

50 marge.java.net/

BY Robert Virkus

Windows Phone

The Ecosystem

Since its introduction in late 2010 the Windows Phone platform has gained high customer satisfaction ratings, even besting the iPhone[1], but that satisfaction has not yielded significant market share. On a global scale, only 2% of all smartphones ran Windows Phone in Q3 2012[2]. Current notable exception is Italy with a market share of more than 10%[3]. The previous high market share in Brazil has declined below 10% over 2012 apparently due to some OEM pricing decisions.

Active Windows Phone vendors are Nokia, HTC, ZTE and Samsung. LG has not released a Windows Phone 8 device.

In Q4 2012, Microsoft introduced Windows Phone 8 which shares a common core with Windows 8 and Windows RT, the desktop and tablet OS. Existing Windows Phone 7 apps will continue to run on Windows Phone 8, but will not be able to access new features and hardware capabilities (unless you use some clever dynamic class loading). Starting with Windows Phone 8 you can also develop apps using C/C++ and DirectX.

In 2012 the amount of available apps in Windows Marketplace has doubled; and the average Windows Phone user now installs 54 apps[4].

1 wmpoweruser.com/q3-2012-survey-finds-windows-phones-outscore-iphone-5-in-customer-satisfaction
2 thenextweb.com/2012/11/01/android-grabs-75-0-market-share-in-q3-followed-by-14-9-for-ios-and-4-3-for-blackberry
3 guardian.co.uk/technology/2012/oct/02/windows-phone-europe-market
4 blogs.windows.com/windows_phone/b/wpdev/archive/2012/12/26/reflecting-on-2012-scale-and-opportunity

Implementation

Windows Phone development is undertaken in C/C++, C# or VB.NET, using the Microsoft Visual Studio IDE or Expression Blend[5]. Applications are created using Silverlight, principally for event-driven applications, and DirectX, principally for games driven by a "game loop", although both technologies can be used in a single application. For Windows Phone 7 you can also create XNA based games. Such games still run on Windos Phone 8, however XNA is not supported for apps that target Windows Phone 8 only. Additionally you can create HTML 5 based apps using PhoneGap[6], however web development is not covered in this chapter.

Metro

Windows Phone's most obvious specific characteristic is the simple-to-use unique interface that focuses on typography and content. This UI paradigm called Metro or Modern UI[7] has been extended to the Xbox 360 and Windows 8 as well. This UI paradigm contains the following principles:

— **Content not Chrome** removes unnecessary ornaments and lets the content itself be the main focus. You should also refrain from using every available pixel, as whitespace gives balance and emphasis to content.

— **Alive in motion** adds depths to the otherwise flattened out design with rich animations

— **Typography is beautiful** moves fonts to first class citizens within Metro. The Helvetica inspired Segoe font of Windows Phone matches the modernist approach.

5 dev.windowsphone.com

6 phonegap.com

7 wikipedia.org/wiki/Metro_(design_language)

— **Authentically digital** design does not try to mimic real world object but instead focuses on the interactions that are available to digital solutions.

You should embrace the Metro/Modern UI design principles in your application, especially when porting over existing apps. Designers will find many inspirations and information in the Microsoft design documentation[8]. One important design aspect is alignment of your UI elements to the standard grid of Windows Phone. Current Silverlight design templates include the grid, you only need to uncomment it in the XAML page source code.

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

New Features

New features of the Windows Phone 8.0 SDK[9] include

— C/C++ support
— DirectX 9_3, XAudio2 and DirectXMath Support
— Interoperability between DirectX/C++ and XAML/C#
— Text to speech
— Speech to text and voice commands
— Lock screen custom counters, icons and background images
— New live tiles sizes and templates
— New Nokia based maps control with offline maps, driving and walking directions
— Better camera app integration with Lenses and auto upload background tasks

8 dev.windowsphone.com/design
9 developer.nokia.com/Community/Wiki/What’s_new_in_Windows_Phone_8

- Easier video recording & more more fine grained camera control
- Wallet integration
- Geo tracking in the background
- Saving calendar entries
- Saving contacts without user prompts
- App to app or web to app communication using custom protocols
- File extension registration
- Access to SD cards
- Low level Bluetooth access; app to app, and app to device communication
- NFC tag reading and writing
- NFC tap and share
- More resolutions: 800x480, 1280x720 & 1280x768
- In app purchase
- Deep integration options for VOIP apps
- Adding and removal of songs
- Social networks: share any media
- Networking: IPv6 and incoming sockets support
- Company app distribution
- Portable class libraries allow easy sharing of code between Windows Phone, Windows 8 and .NET

SDK

The Windows Phone SDK is free of charge and includes "Express" editions of both Visual Studio 2012 and Expression Blend, it needs Windows 8 Pro to run as the SDK uses 'virtualization'. While the Express editions support everything necessary to develop for Windows Phone, many extra features are only available in the commercial editions - most notably the option to create or use portable class libraries. The SDK also includes a device emulator to run code against. The device emulator uses hardware acceleration and features a dashboard for location control, accelerometer simulation and more.

It is important to consider which platform you should leverage when building your application.

Use C# or VB & Silverlight if…	Use C++ & DirectX if…
…you want to create an event-driven application or a casual game.	…you want to create a 2D or 3D game.
…you want to use standard Windows Phone controls.	…you want to manage art assets such as models, meshes, sprites, textures and animations.
…you want to target Windows Phone & Windows 8; re-using lot of code.	…you want to target Windows Phone, Windows 7/8, and Xbox 360; re-using lots of code.

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

Game Engines

With the native app capabilities there are some new game engines for Windows Phone 8:

- Cocos2d-x[10]
- Havok[11]
- Marmalade[12]
- OGRE[13]
- Unity 3D[14]

Services

Push notifications[15] are available that can also update the live tiles of your app. You can also consider using the freely available SkyDrive cloud space and integrate with other Windows Live services[16] for your app.

Multitasking And Application Lifecycle

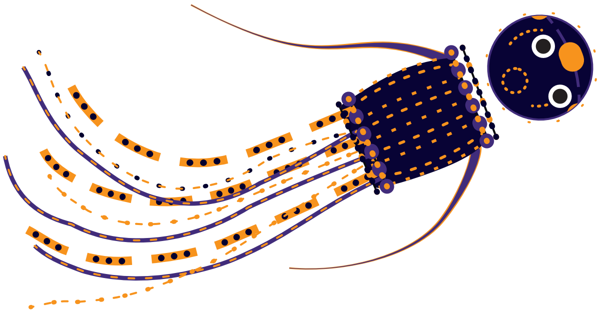
Windows Phone has a limited form of multitasking that suspends applications in the background and allows for fast application switching. The only processes that can be run in the background, after an application has been left, are audio playback, location tracking and file transfer. Applications can also schedule to run arbitrary code in the background at an interval (code which is known as Background Agents). Background Agents are allowed limited

10 cocos2d-x.org/news/76
11 havok.com/products/havok-windows-ecosystem
12 madewithmarmalade.com/windows-phone-8
13 ogre3d.org/2012/10/30/ogre-now-supports-windows-phone-8
14 blogs.unity3d.com/2012/10/30/unity-windows-phone-8-demonstrated-at-microsoft-build-conference
15 msdn.microsoft.com/library/windowsphone/develop/ff402558
16 msdn.microsoft.com/live

use of resources and may be stopped or skipped if the OS determines that the phone needs to conserve resources.

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

To create the appearance of an application that was never closed, Windows Phone has a well-documented application lifecycle called Tombstoning[17]. To make Tombstoning possible, the Windows Phone framework provides the hooks needed to perform actions during different stages of the application lifecycle (such as caching and restoring data and UI states). With Windows Phone 8 there is also a new "fast app resume" feature available to developers.



17 msdn.microsoft.com/library/windowsphone/develop/ff817008

Testing And Analytics

You can unit test applications using the Windows Phone Test Framework[18] or the Silverlight Unit Test Framework[19].

For behavior-driven development, the Windows Phone Test Framework by Expensify[20] is available but this project seems to be in limbo.

For developers wishing to collect runtime data and analytics, there are several options. Localytics[21] and Flurry[22] provide analytics tools and services that are compatible with Windows Phone 7. Developers can also use the Silverlight Analytics Framework[23] to connect to a variety of third-party tracking services such as Google Analytics. Starting with the Windows Phone SDK 7.1 update, there are robust performance monitoring tools available in Visual Studio.

Distribution

Applications for Windows Phone are mainly distributed through the Microsoft Marketplace service. 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 App Hub[24]. Although developer tools are provided free of charge, a paid App Hub account is necessary to deploy software to devices and Marketplace. Currently, a developer account costs 99 USD for an annual subscription and includes 100 free app submissions and unlimited paid

18 wptestlib.codeplex.com
19 nuget.org/packages/WPToolkitTestFx
20 github.com/Expensify/WindowsPhoneTestFramework
21 localytics.com/docs/windows-phone-7-integration
22 flurry.com/flurry-analytics.html
23 msaf.codeplex.com
24 dev.windowsphone.com

app submissions. The fee is waived for students in the Dream-Spark[25] and for the first year for Nokia Publish developers. The Marketplace also provides for time-limited beta distribution and offers a company hub for enterprises[26]. You can use the Windows Phone Marketplace Test Kit[27] to test your application locally before you submit them.

For paid applications, the Windows Phone framework provides the ability to determine if your application is in "trial mode" or not and limit usage accordingly. Microsoft specifically recommends against limiting trials by time (such as a thirty-minute trial) and instead suggests limiting features instead[28].

For ad-based monetization, there are several options. Microsoft has their own Microsoft Advertising Ad Control[29] (currently available in 18 countries), while Nokia[30], Smaato[31], inneractive[32], AdDuplex[33] and Google[34] all offer alternative advertising solutions. For more general information about monetization, please refer to the dedicated chapter in this guide.

25 www.dreamspark.com
26 msdn.microsoft.com/library/windowsphone/develop/jj206943
27 msdn.microsoft.com/library/windowsphone/develop/hh394032
28 msdn.microsoft.com/library/windowsphone/develop/ff967558
29 advertising.microsoft.com/mobile-apps
30 developer.nokia.com/Distribute/NAX
31 smaato.com
32 inner-active.com
33 adduplex.com
34 developers.google.com/mobile-ads-sdk/

Learn More

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

The development team posts on their blog at *windowsteamblog.com/windows_phone* or their Twitter account @wpdev. For a large collection of developer and designer resources, visit *windowsphonegeek.com* and *reddit.com/r/wpdev*.

There are currently several built-in OS controls that are not included in the Windows Phone SDK, such as context menu, date picker, and others. Those controls are available as part of the Silverlight Toolkit for Windows Phone, available at *phone.codeplex.com*. Other popular Windows Phone projects include coding4fun.codeplex.com and *mvvmlight.codeplex.com*. For inspecting the visual tree, bindings and properties of XAML-based user interfaces at runtime, *xamlspy.com* is available.

There are several eBooks available for free, for example Windows Phone Programming in C# (Windows Phone Version 7.5)[35] or Silverlight for Windows Phone Toolkit In Depth[36].

Find many video tutorials at the build conference coverage of channel 9 at *channel9.msdn.com/events/build/2012?t=windows-phone*.

35 blogs.msdn.com/b/uk_faculty_connection/archive/2011/11/23/windows-phone-free-ebook-amp-demos

36 windowsphonegeek.com/WPToolkitBook

BY Robert Virkus

Windows 8

Windows 8 is Microsoft's first OS that runs on tablets and PCs alike. It shares the Modern UI design philosophy with Windows Phone and newer Windows Phone apps will run on any Windows 8 system as well. Windows 8 apps can be developed in C++, C#/VB.NET or JavaScript. They are all first class citizens in the ecosystem as all programming languages have equal access to the Windows Runtime (WinRT) APIs.

While initial Windows 8 adoption was higher in absolute numbers compared to Windows 7, relative adoption has been slower[1]. PC sales continued to decline, however this could be due to relatively few touchscreen models being available. Reception has varied between outward hostile and positively euphoric. In 2012 the Windows 8 store has grown by 35,000 apps within only two months, around the same number of apps that the Windows Phone store acquired in its first year of existence.

Prerequisites

To develop Modern UI style apps you require Visual Studio 12 and Blend, the Express versions are available for free. You can install Windows 8 within a virtual machine, side by side with your existing OS or as your main OS. Having a touch enabled monitor helps to fine tune the user experience for tablets, but Windows 8 work equally well when using a mouse.

Technically you also need a developer license, however this license is automatically acquired and free of charge.

1 phonearena.com/news/Wait-so-Windows-8-is-not-outpacing-Windows-7-adoption-rate_id37212

Implementation

While you may simply choose the language that matches the know-how of you or your team, it is worth understanding the differences in capabilities offered by the various options:

	C/C++	C#/VB.NET	JavaScript
WinRT	yes	yes	yes
Silverlight/ XAML	yes	yes	no
HTML	no	no	yes
DirectX	yes	yes (with SharpDX)	no
Codesharing	Legacy native Windows Apps, professional Xbox, other platforms, …	Legacy .NET Windows Apps, indie Xbox, Windows Phone apps, …	Websites, HTML5 apps, …

If you want to use DirectX with C#, you can use *SharpDX.org, anxframework.codeplex.com* or game libraries based on that like *monogame.codeplex.com.* As Windows Phone 8 and Windows 8 all share the same kernel, it is easy to share code between these platforms. The easiest way for that seems to be the use of shared .NET based libraries between these platforms.

App Parts

Each Windows 8 app consists of several parts:

- **App tile** represents the app on the splash screen and can show relevant content to the user, even when your app is not running;
- **Splash screen** is optionally shown when your app starts;
- **App bar** contains the context relevant actions and commands;
- **Content area** displays your app in different view states such as full screen or snapped, compare the 'Views and Form Factors' section;
- **Charms** allow the user to start interactions with the application, compare the 'Application Contracts' section.

The Windows Runtime APIs

The WinRT APIs are documented on msdn[2], they contain the usual suspects like JSON/XML parsing over geolocation, sensors, media handling and networking APIs. But WinRT has some more rather interesting concepts, for example:

- **Windows.Security.Authentication.Live:** Use Windows Live as an authentication mechanism with zero click single sign-on[3]; share data between a user’s various devices, using SkyDrive[4]
- **Windows.Security.Authentication.Web:** Integrate with web services that use OAuth or OpenID (Facebook, Twitter, etc.)
- **Windows.Security.Credentials:** Enables access to and storage of passwords

2 msdn.microsoft.com/library/windows/apps/br211369
3 msdn.microsoft.com/library/live/hh826544
4 msdn.microsoft.com/library/live/hh826521

- **Windows.ApplicationModel.Contacts:** Access or provide contacts

Application Contracts

Windows 8 features charms in each app that you can access by sliding in from the right hand side. There are five charms: search, share, start, devices and settings. By implementing contracts you can plug into these charms and share information between apps. Declare contracts in your `Package.appx` manifest file, then implement the required functionality.

There are following contracts[5]:

- **Search** searches for content in your app. You can optionally provide search suggestions while the user types. The relevant functionality is found in the `Windows.ApplicationModel.Search` namespace.
- **Share** provides a way to share data between a source and target app, by declaring a corresponding contract. If you want your app to share data, you should make it available in as many data formats as possible to increase the number of potential target apps. You can use standard formats such as text, HTML, images or create your own formats. The share target app can optionally return a quicklink that points to the consumed data. For example, you can share an image with Facebook or Flickr and get back a link. Relevant classes are in `Windows.ApplicationModel.DataTransfer.ShareTarget`.
- **Play To** plays data with connected devices, for example by streaming a video to your DLNA enabled TV. Start with the `Windows.ApplicationModel.PlayTo` namespace.
- **Settings** allows the user to adjust context dependent set-

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

tings from anywhere within your app. Define your settings with the help of `Windows.UI.ApplicationSettings`.

- **App to App Picking** allows you to open or save app files from within another app. Find classes in the `Windows.Storage.Pickers` namespace.

Do not duplicate charms functionality elsewhere in your app. That will just confuse your users. You should, for example, not include a specific search field, unless searching is the main task of your app.

Windows 8 has many more extensions and contracts, a full list is available at *msdn.microsoft.com/library/windows/apps/hh464906*.

Views and Form Factors

Windows 8 apps can run in different layout modes[6]:

- **full screen** is the default mode, either in **landscape** or **portrait** orientation. Your app will use all the available screen real estate to immerse the user completely in `ApplicationLayoutState.FullScreen`.
- **snapped** and **filled** are modes in which apps are shown side by side. You should change your layout accordingly but maintain the state of your app and keep at least the main functions easily acccessible, this applies to both `ApplicationLayoutState.Filled` and `ApplicationLayoutState.Snapped`.

To get notified about layout changes, listen to the `Windows.UI.ViewManagement.ApplicationLayout.GetForCurrentView().LayoutChanged` event. There you can even change the state programmatically:

6 msdn.microsoft.com/library/windows/apps/hh465371

When your app is in snapped mode and your user selects a function that demands a different mode, you can call `ApplicationLayout.TryUnsnap()`.

Autoscaling

Windows 8 runs on devices with different screen resolutions and pixel densities. Depending on the resolution apps are scaled automatically to:

- 1366 x 768 (100%)
- 1920 x 1080 (140%)
- 2560 x 1440 (180%)

Web developers should use SVG graphics and CSS media queries, when possible. XAML developers can use naming schemes for resources, so that the best fitting resource is chosen automatically (such as image.scale-100.jpg, image.scale-140.jpg and image-scale-180.jpg). You should also use resources with dimensions that are multiples of 5px, so that no pixel shifting occurs when autoscaling.

Push

You can send data and even images to your apps using the Windows Notification Service (WNS)[7]. This also enables you to update the live-tiles of your app. Using WNS is free of charge. You can use the Windows Azure Toolkit for Windows 8[8] to simplify the implementation of a push server.

7 msdn.microsoft.com/library/windows/apps/hh465460
8 watwindows8.codeplex.com

Single Sign On

Windows 8 provides user credential management services[9] and using Microsoft Account for its user authentication. You can leverage this to provide single sign to your apps, enabling you to identify the user directly without further authentication[10].

Distribution

Windows 8 apps can be distributed through Windows Store[11] only. The standard revenue share of 70% is increased to 80% when your app makes more than 25,000 USD. The Windows Store will support over 200 countries and regions and more than 100 languages, so you can have a global reach. You also can distribute feature- or time-limited trial versions of your app, use in app purchasing or integrate adverts. Third-party payment providers are allowed as well.

Apps are managed by customer, not by device. So a user can use your app across a variety of platforms, such as a desktop PC and a tablet.

Before you sell apps, you need to obtain a Windows Store account that costs 49 USD per year for individuals and 99 USD for companies.

9 msdn.microsoft.com/library/windows/apps/br211367
10 msdn.microsoft.com/library/windows/apps/hh465097
11 msdn.microsoft.com/library/windows/apps/hh694084

Learn More

Your starting point for Windows 8 development is:
dev.windows.com

Find design tips and tricks at:
design.windows.com.

Discuss development problems on:
social.msdn.microsoft.com/Forums/en-US/category/windowsapps.

Find sample code on *code.msdn.microsoft.com/windowsapps*, in various *codeplex.com* projects and in the end to end samples available at *msdn.microsoft.com/library/windows/apps/br211375*.

The roadmap for app developers provides an good overview about planing, designing and developing Windows 8 apps at: *msdn.microsoft.com/library/windows/apps/xaml/br229583*.

BY Robert Virkus

Going Cross-Platform

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

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

Key Differences Between Mobile Platforms

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

Programming Language

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

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

Language	1st class citizen[1]	2nd class citizen[2]
ActionScript	BlackBerry 10, BlackBerry PlayBook OS (QNX)	none
C, C++	bada, BlackBerry 10, BlackBerry PlayBook OS, Brew MP, Symbian, Windows 8, Windows Phone 8	Android (partially, using the NDK), iOS (partially)
C#	Windows 8, Windows Phone	none
Java	Android, BlackBerry, Java ME devices	Symbian
JavaScript	BlackBerry PlayBook OS, Firefox OS, Tizen, Windows 8	BlackBerry (WebWorks), Nokia (WRT)
Objective-C	iOS	none

1 Supported natively by the platform, for example either the primary or only language for creating applications

2 Supported natively by the platform, for example either the primary or only language for creating applications

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

- **Web Technologies**
- **Interpretation**
- **Cross Compilation**

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[1].

OS Versions

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

UI and UX

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

It is relatively easy to create a nice looking UI that works the same on several platforms. Such an approach, however, might miss important UI subtleties that are available on a single platform only and could improve the user experience drastically. It would also ignore the differences when it comes to the platforms' design philosophies: While iOS strives for a realistic design in which apps look like their real world counterparts, Windows Phone's Metro interface strives for an "authentically digital" experience, in which the content is emphasized not the chrome around it. Another key challenge with a uniform cross-platform UI is that it can behave differently to the native UI users are familiar with, resulting in your application failing to “work” for users. A simple example is not to support a hardware key such as the back key on a given platform correctly. Another challenge is the **uncanny valley** that results

1 www.developergarden.com/apis/

from mimicking native UI elements that look but don't work the same. Instead of mimicking native controls you should either use non-native looking ones or just use the 'real deal'.

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

Desktop Integration Support

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

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

Multitasking Support

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

Battery Consumption And Performance

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

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

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

Push Services

Push services are a great way to give the appearance that your application is alive even when it is not running. In a chat application you can, for example, send incoming chat messages to the user using a push mechanism. The way push services work and the protocols they use, again, can be realized differently on each platform. The available data size, for example, ranges between 256 bytes on iOS and 8kb on BlackBerry. Service providers such as Urban Airship[2] support the delivery across a variety of platforms.

In App Purchase

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

2 urbanairship.com/

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

Cross-Platform Strategies

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

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

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

Component Library	Target Platforms
cocoacontrols.com	iOS
chupamobile.com	Android, iOS

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

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, the design and design assets of the app (but be aware of the need to support platform specific UI constructs).
- **Data structures and algorithms**: Go one step further by sharing data structures and algorithms among platforms.
- **Code sharing of the business model**: Using cross platform compilers you can also share the business model between the platforms. Alternatively you can use an interpreter or a virtual machine and one common language across a variety of platforms.
- **Complete abstraction**: Some cross platform tools enable you to completely abstract the business model, view and control of your application for different platforms.

Player And Virtual Machines

Player concepts typically provide a common set of APIs across different platforms. Famous examples include Flash, Java ME and Lua. This approach makes development very easy. You are dependent, however, on the platform provider for new features and the challenge here is when those features are available on one platform only. Often player concepts tend to use a "least common denominator" approach to the offered features, to maintain commonality among implementations for various platforms. Generator concepts like Applause[3] carry the player concept a step further, they are often domain specific and enable you to generate apps out of given data. They often lack flexibility compared to programmable solutions.

Cross Language Compilation

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

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

3 applause.github.com

(Hybrid) Web Apps

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

Web App Solution	License	Target Platforms
jQuery Mobile www.jquerymobile.com	MIT and GPL	Android, bada, BlackBerry, iOS, Symbian, webOS, Windows Phone
JQTouch www.jqtouch.com	MIT	iOS
iWebKit iwebkit.net	LGPL	iOS
iUI code.google.com/p/iui	BSD	iOS
Sencha Touch www.sencha.com/products/touch	GPL	Android, iOS
The M Project the-m-project.org	MIT and GPL	Android, BlackBerry, iOS, webOS

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

Hybrid web development means embedding a webview within a native app. This approach allows you to access native functionality from within the web parts of your apps and you can also use native code for performance or user experience critical aspects of your app. Hybrid apps allow you to reuse the web development parts across your chosen platforms.

ANSI C

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

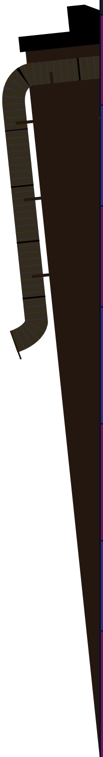
Cross-Platform App Frameworks

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

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

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

Solution	License	Input	Output
Application Craft applicationcraft.com	Commercial	HTML, CSS, JavaScript	Android, Black-Berry, iOS, Symbian, Windows Phone, mobile sites
appMobi appmobi.com	Commercial	HTML, CSS, JavaScript	Android, iOS, Kindle Fire, Nook, web
Codename One codenameone.com	Commercial	Java	Android, BlackBerry, iOS, J2ME, Windows Phone
Corona coronalabs.com (Corona Labs)	Commercial	JavaScript	Android, iOS
J2ME Polish j2mepolish.org (Enough Software)	Open Source + Com-mercial	Java ME, HTML, CSS	Android, BlackBerry, J2ME, PC
Flash Builder adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, BlackBerry Playbook OS, iOS, PC
Feedhenry feedhenry.com	Commercial	HTML, CSS, JavaScript	Android, BlackBerry, iOS, Windows Phone
Kirin/JS kirinjs.org/	Open Source	JavaScript	Android, iOS
Kony One kony.com/node/4	Commercial	HTML, CSS, Ja-vaScript, RSS	Android, BlackBerry, iOS, J2ME, Symbian, Windows Phone
LiveCode runrev.com (RunRev)	Commercial	English-like	Android, iOS, PC and Web



Solution	License	Input	Output
MobiForms www.mobiforms.com (MobiForms)	Commercial	Drag and Drop + MobiScript	Android, iOS, PC, Windows Mobile
Mono for Android xamarin.com/ monoforandroid (Xamarin)	Commercial	C#	Android (share code with iOS and Windows Phone)
Mono Touch xamarin.com/ monotouch (Xamarin)	Commercial	C#	iOS (share code with Android and Windows Phone)
MoSync mosync.com	Open Source + Com-mercial	C/C++, HTML5/JS	Android, BlackBerry, iOS, J2ME, Symbian, Windows Phone 7, Windows Mobile
NeoMAD neomades.com	Commercial	Java	Android, bada, BlackBerry, iOS, J2ME, Symbian, Windows Phone 7
PhoneGap/Cordova www.phonegap.com (Adobe/Apache)	Open Source	HTML, CSS , JavaScript	Android, Black-Berry, iOS, Symbian, Windows Phone
Qt qt.digia.com (Digia)	Open Source + Com-mercial	C++	PC, Symbian, MeeGo and Windows Mobile, desktop Windows, Apple & Linux OS
Rhodes rhomobile. com/products/rhodes (Motorola)	Open Source + Com-mercial	Ruby, HTML, CSS, JavaScript	Android, Black-Berry, iOS, Symbian, Windows Mobile, Windows Phone

Solution	License	Input	Output
Spot Specific www.spotspecific.com	Commercial	Drag and Drop, JavaScript	Android, iOS
Titanium www.appcelerator.com (Appcelerator)	Open Source	JavaScript	Android, Consoles, iOS, PC
trigger.io trigger.io (Triggger Corp)	Commercial	HTML5, JavaScript	Android, iOS, Windows Phone
Verivo verivo.com	Commercial	(Visual)	Android, BlackBerry, iOS
webMethods Mobile Designer (formerly Metismo Bedrock) www.metismo.com (Software AG)	Commercial	Java ME	Android, bada, BlackBerry, brew, Consoles, iOS, PC, Windows Phone, Windows Mobile
XML VM xmlvm.org	Open Source + Commercial	Java, .NET, Ruby	C++, Java, JavaScript, .NET, Objective-C, Python

Cross-Platform Game Engines

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

Solution	License	Input	Output
Cocos 2D cocos2d-x.org	Open Source	C++, HTML5, JavaScript	Android, BlackBerry, iOS, Windows 8, Windows Phone
Corona coronalabs.com (Corona Labs)	Commercial	Lua	Android, iOS, Kindle, nook
EDGELIB edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC, Symbian
Esenthel esenthel.com (elements interactive)	Commercial	C++	Android, iOS, PC
GameSalad gamesalad.com	Commercial	Drag and drop	Android, iOS, PC, web
id Tech 5 idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC
Irrlicht irrlicht.sourceforge.net	Open Source	C++	Android & iOS with OpenGL-ES version, PC
IwGame drmop.com/index.php/iwgame-engine	Open Source	C++	Android, bada, BlackBerry Playbook OS, iOS, PC

Solution	License	Input	Output
Marmalade madewithmarmalade.com (Ideaworks3D)	Commercial	C++, HTML5, JavaScript	Android, bada,BlackBerry 10, BlackBerry PlayBook OS, iOS, LG Smart TV, Windows Phone
Moai getmoai.com (Zipline Games)	Commercial	Lua	Android, iOS, PC, Web
MonoGame monogame.codeplex.com	Open Source	C#, XNA	Android, iOS, PC, Windows 8
Ogre 3D ogre3d.org	Open Source	C++	Windows 8, Window Phone, PC
orx orx-project.org	Open Source	C, C++, Objective-C	Android, iOS, PC
ShiVa 3D stonetrip.com	Commercial	C++	Android, BlackBerry 10, iOS, PC, Consoles
SIO2 sio2interactive.com (sio2interactive)	Commercial	C, Lua	Android, bada, iOS, PC
Unigine unigine.com (Unigine corp.)	Commercial	C++, Unigi-neScript	Android, iOS, PC, PS3
Unity3D unity3d.com (Unity Technologies)	Commercial	C#, JavaScript, Boo	Android, BlackBerry 10, iOS, Windows Phone, PC, consoles, web

MOBILE

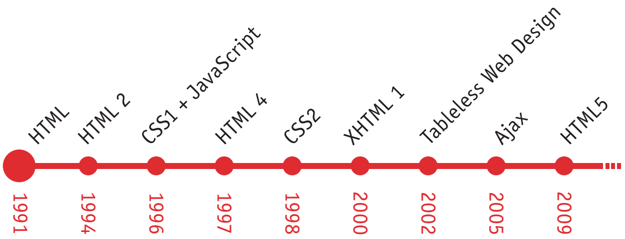
BY Daniel Kranz

Web Technologies

In 2010 Mary Meeker predicted that mobile Internet traffic would surpass desktop usage within 5 years[1]. Her recent report[2] is well worth reading. 2012 saw mobile Internet traffic grow from 8% to 13%. Particular countries have seen an explosion of mobile Internet traffic. As of mid 2012, India is one of the first countries where mobile Internet traffic has already surpassed desktop Internet usage[3].

Continuous web technology development coupled with an increase of internet-capable devices promises a great future for people catering to the ever-increasing mobile web audience.

Rough timeline of web technologies[4]:



One big advantage of web technologies is that they offer the easiest route into mobile development. For a web developer, mobile is simply part of the web. Web technologies, such as HTML, CSS and JavaScript have already been highly developed

1 gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet/
2 www.businessinsider.com/mary-meeker-2012-internet-trends-year-end-update-2012-12
3 gs.statcounter.com/#mobile_vs_desktop-IN-monthly-201111-201211
4 slides.html5rocks.com/#timeline-slide

for many years; however they remain, and will continue to be, the main drivers of mobile site development. Additionally, they are arguably also easier to learn than some of the rather complex native languages needed for native app development. Mobile websites and web apps make content accessible on almost any platform with less effort in comparison to native development for a number of platforms. This means mobile websites automatically have a wider reach. Accordingly mobile web development not only saves development time and cost, but furthermore provides a time and cost-effective alternative when it comes to maintenance. And being independent from appstores allows you to offer any content you want quickly, and without having to align it to the appstore'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 and monetization of mobile sites can prove tricky since users expect to access mobile sites free of charge. The most common monetization tool for mobile sites is ad integration. Payment solutions for mobile sites are still in its early stages and tend to be rather challenging to implement. Existing app store monetization tools on the contrary offer an easy set-up and a high level of security for the end-user.

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

HTML5

The fifth version of the HTML standard promises the reproduction of features previously only available with the help of proprietary technology. HTML5 is one of the key drivers that make developers consider developing mobile sites 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 hardware of devices such as the camera and microphone, data storage on devices to operate mobile sites offline and optimization of web page displays based on screen size make HTML5 an appealing alternative to native app development.

However HTML5 relies on browser support and exactly that support is currently lacking. Only 60% of Internet users have browsers that support more than 50% of HTML5's current features[5].

Ex-Facebook CTO Brent Taylor describes the situation as follows:

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

Mark Zuckerberg even went a step further, naming Facebook's HTML5 app 'one of the biggest mistakes if not the biggest strategic mistake' they made[6].

5 gs.statcounter.com/
6 news.cnet.com/8301-1023_3-57511142-93/html5-is-dead-long-live-html5

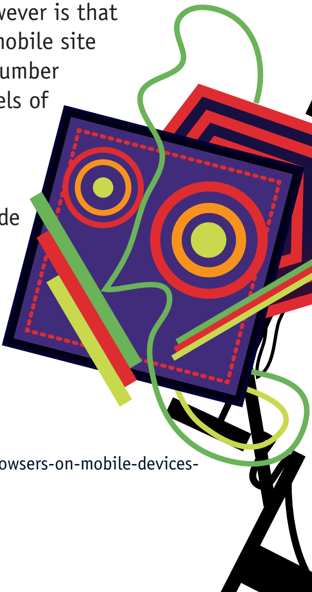
Nevertheless, both Taylor and Zuckerberg believe in HTML5 in the long run. Facebook has also launched ringmark[7] which tests web browsers for 3 rings, or levels, of support for HTML5 features which helps developers to quickly check the level of support of various mobile (and desktop) web browsers.

ABI Research estimates that while in 2010 only 109 million devices offered a browser with some sort of HTML5 support this number is bound to increase to 2.1 billion mobile devices by 2016[8]. Furthermore, the Worldwide Web Consortium (W3C) has finally declared HTML5 feature complete and envisions that HTML5 will be an official web standard by 2014[9].

Fragmentation Needs Adapation

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

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



7 rng.io/

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

9 www.w3.org/

- 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 it 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

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

Unfortunately responsive design can only provide a device-sensitive experience to a limited range of devices and lacks sophisticated content adaptation. The same content is served to all devices. It is not advisable as a technique to deliver complex desktop and mobile sites.

Pro:

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

Con:

- The same content available on the web site will also be available on the mobile version (visible or not).

- Pageweight of the site can have a significant impact in terms of performance on mobile devices
- It is a general approach instead of actual mobile-friendly device optimization (e.g. Top 5)

Progressive Enhancement

Progressive Enhancement has the capability to cater to the full spectrum of mobile devices. A single HTML page is sent to every device. JavaScript code is additionally 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[10].

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 client-side impacts overall performance of the site
- The same HTML page is served to all devices

10 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[11] and DeviceAtlas[12]. The main drawback of device databases is that the majority is only available as part of a commercial license.

Pro:

- Most commonly used solution (Google, Facebook, Amazon...)
- Maximum control
- Device optimization possible (eg. iPhone, Samsung Galaxy III, ...)

Con:

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

Hybrid Adaptation

Truly the best of both worlds, the combination of client and server-side adaptation ensures high performance thanks to server-side adaptation and ensures that capabilities sourced can be used to enrich the mobile experience on subsequent visits.

11 wurfl.sourceforge.net/
12 deviceatlas.com/

Hybrid adaptation solutions are available commercially from companies like Sevenval[13] or Netbiscuits[14], or as community-backed cloud solutions, e.g. FITML[15] .

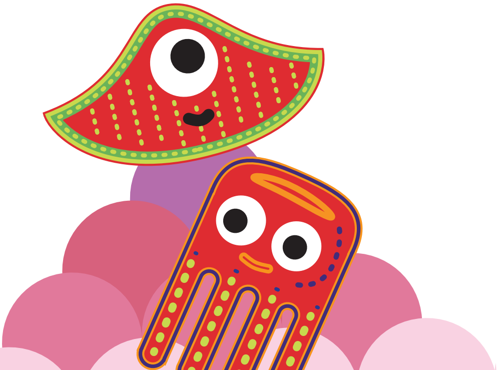
Better Data Input

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

Testing Web Technologies

How web technologies work in various mobile phones can be tested in several ways. The simplest way is to to try testing the web site or web app in a variety of web browsers on

13 www.sevenval.com
14 www.netbiscuits.com
15 www.fitml.com



mobile devices. These would include a mix of the most popular mobile web browsers, for example based on public data *gs.statcounter.com/#mobile_browser-ww-monthly-201111-201211*. The set of devices can be refined by analyzing data from existing web logs, et cetera. Also, testing on various form-factors helps to expose layout and formatting issues.

In terms of automated testing, WebDriver[16] 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 other 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 e.g. the Nokia Asha 201 phone. However the behaviour and rendering is not realistic so many bugs will remain latent, while other **false positive** bugs will be found that do not actually happen on real devices.

16 seleniumhq.org/projects/webdriver/

Learn More

Online

- **HTML5 Rocks:**
  Great resource about HTML5 including tutorials, slideshows, articles, etc. *www.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 *www.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
- **WHATWG:**
  The HTML community's homepage *www.whatwg.org*
- **Word Wide Web Consortium:**
  The organization that defines web standards *www.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 Gary Readfern-Gray & Julian Harty

Accessibility

Regardless of the technology you choose to develop your apps, you will want to ensure that your app can be used by as many people in as many different markets as possible.

Many of your potential users may have a disability which makes it more difficult for them to use mobile technology. These disabilities include, but are not limited to, various levels of sight or hearing impairment, cognitive disabilities, dexterity issues, technophobia and the like. Many of these users rely on third-party applications such as TalkBack on the Android platform or Talks from Nuance for the Symbian platform, which provides screen reading and screen magnification features. iOS now includes VoiceOver[1] which is the front-runner in terms of providing an accessible interface on mobile phones.

To make your software accessible for users with disabilities, you should follow some general guidelines. If you stick to them, you will also give your app the best chance of interoperating with any third-party access software that the user may be running in conjunction with your software:

- 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
- Follow the standard UI guidelines on your platform. This enhances consistency and may mean a more accessible design by default

1 www.apple.com/accessibility/iphone/vision.html

- Label all images with a short description of what the image is, such as “Play” for a play button.
- Avoid using colour as the only means of differentiating an action. For example a colour-blind user will not be able to identify errors if they are asked to correct the fields which are highlighted in red.
- Ensure good colour contrast throughout your app.
- 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.
- 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.
- Test your app on the target device with assistive technology such as VoiceOver on the iPhone.

You can find a more comprehensive list of guidelines online[2].

Apple and more recently Google, 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, which is 'a good thing'.

Looking at the different mobile platforms more closely, it becomes obvious that they differ largely regarding their accessibility features and APIs.

2 www.slideshare.net/berryaccess/designing-accessible-usable-application-user-interfaces-for-mobile-phones

Accessible Android Apps

The latest major version of Android, Version 4, brings a raft of accessibility improvements. These include the accessibility focus, Braille support and more. The developer documentation has also been enhanced. However, to maximise the reach of your app to those using previous versions of Android, you should use standard UI controls where possible and make sure users can navigate your app via a trackball or D-pad. This will give your app the best chance of being rendered accessibly by the likes of Talkback and other assistive technology applications.

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[3].

You will also find more examples in the training area of the developer documentation in a section entitled Implementing Accessibility[4]. Testing the Accessibility is also covered online[5].

For more information about Android accessibility including how to use the text to speech API, see the Eyes-Free project[6].

3 developer.android.com/guide/topics/ui/accessibility/apps.html
4 developer.android.com/training/accessibility/index.html
5 developer.android.com/tools/testing/testing_accessibility.html
6 code.google.com/p/eyes-free

Accessible BlackBerry Apps

BlackBerry also provides good and extensive information about the use of their accessibility API and many hints on accessible UI design on their website for developers[7].

In May 2012 Blackberry Released the BlackBerry Screen Reader[8] for various recent BlackBerry® Curve™ smartphones. This is available as a free download which you may wish to use in the testing of the accessibility of your apps.

Accessible iOS Apps

iOS has good support for accessibility. For example, iOS devices include:

— **VoiceOver** a screen reader. It speaks the objects and text on screen, enabling your app to be used by people who may not be able to see the screen clearly
— **Zoom** This magnifies the entire contents of the screen
— **White on Black** This inverts the colors on the display, which helps many people who need the contrast of black and white but find a white background emits too much light
— **Captioning and subtitles** for people with hearing loss
— **Audible, visible and vibrating alerts** to enable people to choose what works best for them
— **Voice Control and Siri** This enables users to make phone calls and operate various other features of their phone by using voice commands.

7 https://developer.blackberry.com/java/documentation/intro_accessibility_1984611_11.html
8 www.blackberry.com/screenreader

If you are working on iOS, make sure to follow Apple's accessibility guidelines[9]. These guidelines detail the API and provide an excellent source of hints and tips for maximising the user experience with your apps.

Accessible Symbian / Qt Apps

At the time of writing, there is no “accessibility API” for the Symbian platform, however there are several third party apps that provide good access to many Symbian phones along with many of the apps they use.

When developing native Symbian apps your best chance of developing an accessible app is to use the standard UI controls where possible. If you are developing using Qt, then please check the web for details of their accessibility API[10].

Accessible Windows Phone & Windows 8 Apps

As with the other major mobile platforms, the accessibility features of Windows Phone are being enhanced with every successive release.

As you probably learned in the Windows Phone and Windows 8 chapter, you essentially have two choices when writing apps for the platform.

If your app is written in C# C++ or Visual Basic, you will find comprehensive information on making your app accessible in the document Accessibility in Metro style apps using C++, C#, or Visual Basic[11].

9 developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility

10 doc.qt.nokia.com/qq/qq24-accessibility.html

11 msdn.microsoft.com/en-us/library/windows/apps/xaml/hh452680.aspx

If you have chosen to use HTML 5 and JavaScript, then you will need Accessibility in Metro style apps using JavaScript[12].

Once you have tested the accessibility of your app[13], Microsoft uniquely allows you to declare your app as accessible[14] in the Windows store, allowing it to be discovered by those who who are filtering for accessibility in their searches.

Accessible Mobile Web Apps

Much has been written on the subject of web accessibility, however, at the time of writing, there is no standard which embodies best practice for accessible mobile web development.

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

If you are a web content developer, then you should take a look at the Web Content Accessibility Guidelines (WCAG) Overview[15].

As support of HTML 5 is increasingly adopted on the various mobile platforms, you might find it useful to take a look at the document entitled Mobile Web Application Best Practices[16] as this is likely to form the foundation of any mobile web application accessibility standard that emerges in the future.

You will also find Relationship between Mobile Web Best Practices (MWBP) and Web Content Accessibility Guidelines (WCAG)[17] a helpful resource.

12 msdn.microsoft.com/en-us/library/windows/apps/hh452702.aspx
13 msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994937.aspx
14 msdn.microsoft.com/en-us/library/windows/apps/xaml/jj161016.aspx
15 w3.org/WAI/intro/wcag
16 w3.org/TR/mwabp
17 www.w3.org/TR/mwbp-wcag/

BY Ian Thain & Davoc Bradley

Enterprise Apps, Strategy And Development

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

It may seem an obvious thing to say, but the major risk at the moment, is **not** having an enterprise mobile strategy. Business is now looking at **Mobile for All** rather than limiting it to senior management, as it may have been in the past. To enable this the traditional IT approach of buying devices and distributing them throughout the management structure is no longer the only enabling strategy being used; Bring Your Own Device (BYOD) is taking hold, enabling staff to use their personal devices to connect to the IT infrastructure, download secure content and use enterprise apps. With the advent of BYOD, 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.

Key points for Mobile Apps in Shaping the new Business Enterprise are:

- 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

Strategy

Many companies nowadays have a Chief Mobile Officer (CMoO) or equivalent position. It is the CMoO's 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 CMoO might consider setting up a Mobile Innovation Council (MIC). The MIC 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 needs to focus on together with the MIC include:

- **Strategy:** Vision and direction for the general mobile strategy and for the apps.

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

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

Key points when building the business case for Mobile Enterprise Apps are:

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

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).

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

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

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

- Device monitoring
- License control
- Distribution via an internal Over-The-Air (OTA) solution
- Software inventory

- Asset control
- Remote control
- Connection management
- Application support & distribution

Security measurements include:

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

Although MDM and MAM solutions offer app management and delivery to devices, providing cross platform management, there are also operating system specific methods for some aspects of app management, delivery and deployment.

- Android provides the Google Play Store for Enterprise which allows a company to distribute and manage apps for Android through a customized secure Google Play Store
- Apple provides the Volume Purchasing Agreement, which allows companies to bulk purchase public App Store apps to be distributed to their staff, allowing staff to download those apps free until the number pre-purchased has been downloaded
- Microsoft provides the Microsoft System Center 2012 (formerly System Center Mobile Device Manager), which enables MDM for Windows devices
- RIM BlackBerry provides the Business Enterprise Server (BES), enabling high detail MDM for BlackBerry devices

Mobile Enterprise Application Platforms (MEAPs)

Usually, one key element of enterprise applications is data synchronization. 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 Enterprise Application Platform (MEAP) solutions. MEAPs 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 synchronization. Available solutions include:

- Amp Chroma by Antenna[1]
- Kony[2]
- SpringWireless[3]
- Sybase Unwired Platform[4]
- Syclo[5]

1 www.antennasoftware.com
2 www.kony.com
3 www.springwireless.com
4 www.sybase.com/products/mobileenterprise/sybaseunwiredplatform
5 www.syclo.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.

Key points for securing Enterprise Apps:

- 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 Julian Harty

Mobile Analytics

Our apps are used remotely by people we may never meet. Mobile Analytics can help us to discover how your app is being used so we can improve future releases of the app. Over half of the top mobile apps already include mobile analytics[1].

There are lots of commercial offerings that claim they provide powerful, pain-free solutions to add mobile analytics to our apps. There are even a few opensource offerings. We could simply pick one of these solutions and hope they will work for us. However, we risk drowning in a turbulent sea of data where we do not understand what the data means and where we cannot easily extricate ourselves. This chapter includes some tips and guidance to help you understand how mobile analytics can help you understand how your app is being used. You will discover how to pick an appropriate solution and to implement it into your app.

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

Getting Started

Many providers of mobile analytics solutions offer a 'Getting Started' section where you learn how to take your first step when using their products. Examples include Flurry[2] and KISSmetrics[3].

Generally you need to register before you can usefully use any of the products as they need configuring with a unique 'key' for your app.

Consider several of the potential solutions before committing to any of them. Read the documentation and example code to see how easily you can implement it 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.

You should be aware that most mobile analytics solution providers are also extracting and using the 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. Also if there is personally identifiable information in the data, there may be additional legal and privacy implications. So it is worth considering how third-parties will use and share the data reported via their software and APIs.

2 support.flurry.com
3 support.kissmetrics.com/getting-started/overview

Deciding What To Measure

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

- **Key usage events:** What are the key features of your app mostly used?
- **Business-centric events:** Any interaction of the user that generates revenue for you: How often do your users purchase the premium version of your app or other items offered within your software? When do they cancel orders or discard their shopping cart before checking out?
- **Usability metrics:** Where do your users get stuck in the usage flow? Do they quickly reach their goal when using your software?

Once you 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, rather than simply numbering them, 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.

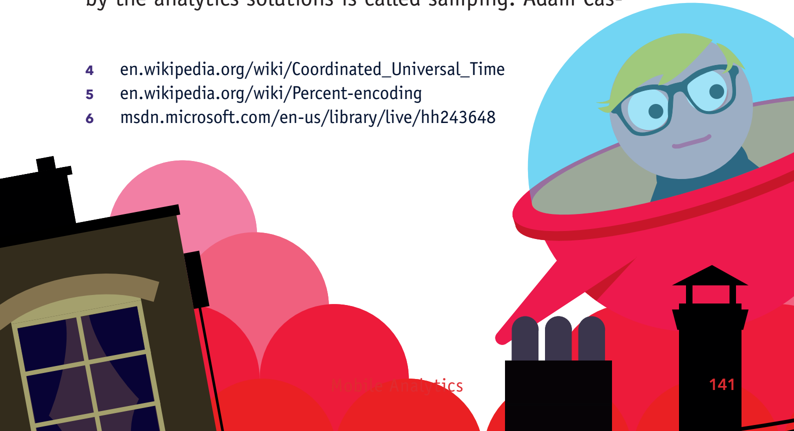
Also remember to address globalization issues such as the timestamp of each element. Does the app detect the time of an event according to the device's location, the device's settings or does it use a global time like UTC time[4]?

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

Custom event tags augment predetermined events, and many of the mobile analytics solutions provide ways for your app to generate them. You may need to format the custom event messages. If so, pay attention to encoding of the elements and separators; for instance they may need to be URL encoded[5] when they are sent as REST messages[6] .

You may want to think about how often the app should report events and reduce the risk of flooding the available capacity of the analytics system, which might affect the reliability and accuracy of the delivered analytics data. One method to reduce the volumes of data processed by the analytics solutions is called samping. Adam Cas-

4 en.wikipedia.org/wiki/Coordinated_Universal_Time
5 en.wikipedia.org/wiki/Percent-encoding
6 msdn.microsoft.com/en-us/library/live/hh243648



sar published an interesting blog post on this topic at *www.periscopix.co.uk/blog/should-you-be-worried-about-sampling/*.

Adjusting Your Code

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

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

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

For Windows Phone, capabilities are used to decide what the app uses. Localytics has a quickstart guide online[7] that includes an example of setting the `ID_CAP_IDENTITY_DEVICE` capability.

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

Handling the results

Be aware that 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 a day or longer. You, and your business sponsors, need to decide how long you can afford to lag the real-time events.

Some analytics solutions also provide an API to allow you to access the data. This may enable you to make backups, and to create custom reports.

To evaluate the quality of the results, some companies invest to 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[8].

If you have decided to work with KISSmetrics, check out their article on ways to test your metrics at *support.kissmetrics.com/getting-started/testing-km*.

Privacy

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.

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

The providers of third-party libraries seem to have a range of attitudes to privacy. Some claim the privacy of users is paramount and stresses the importance of not tracking users. Google Analytics clearly prohibit tracking personally identifiable information in their terms of service[9]. 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 `sharedAPI] identify:@"name@email.com"];`[10]. And mixpanel provides an example of how to update a user's People Analytics record[11]

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[12]
- Kord Davis' book "Ethics of Big Data"

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:

- The Mobile Developer's Guide to the Parallel Universe, a sister book to this one, covers Mobile Analytics from a marketing perspective. Available as a pdf download at *www.wipconnector.com*
- "Mobile Analytics" by Jesus Mena is available as a Kindle book via Amazon

9 www.google.com/analytics/terms/us.html
10 support.kissmetrics.com/apis/objective-c
11 mixpanel.com/docs/people-analytics/android
12 jnorthrop.me/2012/07/2/privacy-considerations-mixpanel-people-analytics/

BY Michel Shuqair

Implementing Rich Media

“As many standards as handsets” is a truism when it comes to the list of supported media formats on mobile phones. In contrast to PCs, where most audio and video formats are supported or a codec can easily be installed to support one, mobiles are a different story. To allow optimization for screen size and bandwidth, specific mobile formats and protocols have been developed over the past few years. Small variations in resolution, bit rate, container, protocol or codec can easily cause playback to fail, so always test on real devices.

That said, most of today's smartphones support MP4 h.264 320x240 AAC-LC, however multiple variations are possible among handsets, even within one vendor or firmware version. New formats are still added every year, such as WebM/vp8[1], an open video standard running on Android 4+ in an attempt to become the HTML5 standard (but not supported by Apple yet).

1 en.wikipedia.org/wiki/VP8

Below are the recommended full screen formats for highest compatibility:

Container	mp4, 3gp, avi (BlackBerry only), wmv (Windows Phone + BB10 only)
Protocol	HTTP (progressive or download) or RTSP (streaming)
Video	H.264, H.263
Audio	AAC-LC, MP3, AAC+
Classic Resolutions	176x144 (Older phones), 320x240, 480x320 (J2ME)
Common Resolutions	480x800, 640x480 (Blackberry), 960x640 (iPhone), 1024x768 (iPad 1+2), 2048x1536 (iPad 3+4)
HD Resolutions	1280x720 (BB10, Samsung, Windows Phone 8), 1136x640 (iPhone 5)

Streaming vs. Local Storage

There are two options to bring media content to mobile devices: Playing it locally or streaming it in real time from a server.

To stream content through relatively unstable mobile networks, a specific protocol called RTSP was developed that solves latency and buffering issues. Typical frame rates are 15 fps for MP4 and 25 fps for 3gp, with data rate up to 48 kbps for GPRS (audio only), 200 kbps for Edge, 300 kbps for 3G/UMTS/WCMDA and 500 kbps for Wi-Fi and 4G. HD-video starts at 2Mbps and is not recommended for streaming (yet).

Apple's open source Darwin streaming server[2] can serve streaming video and audio with the highest level of compatibility and reliable RTSP combined with FFMPEG[3] and is always a good choice to stream 3gp or mp4 files.

When targeting Windows Mobile/Phone, Windows Media Server[4] is preferred to support HTTP streaming. Android 3.0 upwards also supports HTTP streaming. Note that atomic hinting is required (see Progressive Download) and mp4 files are very strict in encoding (use H.264 15 fps AAC-LC 48khz stereo). Only HTC Android devices and Android 4.0 devices are less strict in streaming formats and will play much more encoding variations than other brands.

When streaming is not available on the phone, blocked by the carrier or you want to enable the user to display the media without establishing a connection each time, you can of course simply link and download the file. This is as easy as linking to a download on the regular web, but mobile phones might be stricter in checking for correct mime types. Use audio/3gp or video/3gp for 3gp files and video/mp4 for mp4 files.

Some handsets simply use the file extensions for data type detection, so when using a script – such as download.php – a well-known trick is to add a parameter such as `download.php?dummy=.3gp` to ensure correct processing of the media. Some phones cannot play 3gp audio without video, but a workaround is to include an empty video track in the file or a still image of the album cover.

Depending on the extension and protocol, different players might handle the request. On some phones, like Android, multiple media players can be available and a popup is displayed to allow the user to select one.

2 dss.macosforge.org
3 www.ffmpeg.org
4 www.microsoft.com/windows/windowsmedia/forpros/server/server.aspx

Finally you can simply include media files in your mobile app as a resource. On Android devices pay attention to support media located on the SD-Cards (Android 3.1 and up) which requires the `android.permission.READ_EXTERNAL_STORAGE` permission.

Progressive Download

To avoid configuring a streaming server, a good alternative is to offer progressive downloads, for which your media files can be served from any web server. To do this, you have to **hint** your files. Hinting is the process of marking several locations in the media, so a mobile player can start playing the file as soon as it has downloaded a small part of it (typically the first 15 seconds). Note: an mp3 file does not need hints, and cannot be hinted.

Possibly the most reliable open source hinting software available is Mp4box[5].

Media Converters

To convert a wide variety of existing media to mobile phone compatible formats FFMPEG is a must have (open source) media format converter. It can adjust the frame rate, bit rate and channels at the same time. Make sure you build or get the binary with H263, H264, AAC and AMR encoder support included. There are good converters available based on FFMPEG, such as “Super” from eRightSoft[6]. For MAC users, QuickTime pro (paid version) is a good alternative to encode and hint 3gp and mp4 files. If you are looking for a complete server solution with a Java / opensource background, check out Alembik[7].

5 gpac.wp.institut-telecom.fr/mp4box/
6 www.erightsoft.com/super
7 www.alembik.sourceforge.net

BY Alex Jonsson

Implementing Location-Based Services

Location based services are one of the hot areas for mobile applications. While nobody has yet proven that offering position and heading information to be lucrative in itself, apps which contain a geographically-aware component lead to more relevant services, which in turn may contribute to greater revenue. Knowing a user's location means you can deliver more relevant information; helping them find a nearby restaurant taking into account the local weather forecast, finding where friends are, or helping users find most scenic local bike routes as crowdsourced by other bikers. Yet getting location data is only half the story, providing the user with a meaningful representation is a key factor in many apps, which usually means delivering a graphical representation overlaid with routes, points-of-interest, et cetera. Yet, a comprehensive list of resources assorted by proximity, can many times be more fruitful than a scrollable, slow map view.

In short; location-based services often work silently in the background and are not always transparent towards the end-user.

How To Obtain Positioning Data

Location-based applications can acquire location information from several sources; via one of the phone's available network connections, GPS satellites, short range systems based on visible tags or local short range radio, or old-school by inputing data via the screen or keyboard.

- **Network positioning:** Each GSM or UMTS base station carries a unique ID, containing its country code, network id, five-digit Location Area and two-digit Routing Area. The coordinates of a base station can then be obtained by looking up the operator's declaration in a database. This information is not particularly accurate in terms of pin-pointing our exact location and depends on the cell size (base station coverage): Cells are placed more densely in urban areas which provides greater accuracy than in rural areas. Techniques, such as measuring the difference in the time-of-arrival of signals from several nearby base stations (known as multilateration) can help improve accuracy, while telephony providers charge for these premium network services. For phones with WiFi capabilities, lists of known wireless LAN access points are used by companies, including Google.
- **GPS positioning**: An on-board GPS module (or an external one) typically gives you a 50% accuracy ranging from 5 to 50 meters, depending on the quality of the hardware and how many satellites the GPS module finds in the sky at any given time. Accuracy is also affected by the terrain, canopy and wall materials; any of these may obscure the satellite signals: In cities, urban canyons created by clusters of tall buildings can distort the signal, giving false or inaccurate readings. Combining GPS with network positioning is

increasingly common: Assisted GPS, or A-GPS, uses an intermediary, called an Assistance Server, in order to minimize the delay to the first GPS fix. The server uses orbital data, accurate network timing and network-side analysis of GPS information. However, A-GPS does not mean a more accurate position, but rather a faster result when the GPS is initially enabled, or when GPS satellite coverage is poor. This shortens the time needed for a location lock. Note: most A-GPS solutions require an active cellphone network connection.

- **Short range positioning**: Systems based on sensors; near field communication (NFC), Bluetooth and other radio-based tag systems – use active or passive sensors in proximity to points of interest, such as exhibits in a museum or stores in a shopping mall. Low-tech solutions include bar codes and other visual tags (such as QR codes) that can be photographed and analyzed on a server or locally on the phone; such tags may contain an id from which a position can be looked up. The user can specify their position by selecting a location on a map, inputting an area code or a physical address. This option is used typically for applications on feature phones, which may lack other means of determining a location.

Mapping Services

Few people missed the less-than-successful introduction of Apple's maps in second half of 2012, where cities, landmarks and roads were missing or incorrect; and the Google's launch of a map application for iOS right before Xmas.

In general, a map service takes a position and a bunch of metadata as input parameters and returns a map, also layered with contextual metadata. The map itself can be in the form of one or more image bitmaps, vector data or a combination of both. Vector data has several advantages over bitmaps: vector representations consumes less bandwidth and enables arbitrary zooming from space into your face. However it requires more processing on the client side. Bitmaps are often provided in discrete zoom levels, each with a fixed magnification, named after its coordinates.

Free maps, both served as bitmaps and vectors, include Open Street Map[1] or CloudMade[2]. Commercial maps include Garmin[3], Microsoft's Bing resources[4] to name a few.

Some solutions, such as Google Maps[5], are free when your application is made available at no cost, but require you to obtain a map key. Other map services, such as Google's static maps, are limited to serving a number of tiles to a map key or IP address. Several of the sources share similar map formats and are thus interchangeable.

Implementing Location Support On Different Platforms

Location API for Java ME offers detail such as the latitude and longitude position, the accuracy, response time, and altitude derived from the on-board GPS as well as speed based on performing consecutive readings.

With iOS there is integrated support for location but with restrictions on how the location data can be generated by the supporting functions. Currently, there is also an on-going debate on how location data is recorded and stored on the iOS devices and how Apple are planning to use this data for

1 wiki.openstreetmap.org/wiki/Software
2 www.cloudmade.com
3 garmin.com
4 www.microsoft.com/maps/developers
5 code.google.com/apis/maps

their own purposes. Android developers also have access to high-level libraries and these devices are more liberal with the choice of map sources, although they default to Google's map APIs. On Symbian devices, Nokia Maps can be used free of charge including commercial use.

Since iOS 3.x and Android 2.0, Web app developers have been able to access geoinformation via the navigator.geoposition interface, e.g calling `navigator.geolocation.getCurrentPosition(my_geo_handle)` gives you the opportunity to fetch the `my_geo_handle.coords.latitude` and `my_geo_handle.coords.latitude`, after given permission from the user and satellites are available. Via clever scripting, this action can be combined with fallbacks to network lookups.

Geographical data often is presented with other information, available in a number of formats. One of the widely accepted standards is called geoRSS, and could look like this for a single point-of-interest:

```
<entry>
<title>Byviken's fortress</title>
<description>Swedish 1900-century army
installation, w. deep mote
</description>
<georss:point>18.425 59.401</georss:point>
</entry>
```

There are other formats for geodata, but the basic idea is similar; by harmonizing data streams and webservices, robust mashups can be created to run seamlessly in various user contexts. Other important formats for geoinformation include the Geography Markup Language (GML), an XML encoding specifically for the transport and storage of geographic information, and KML which is an elaborate geoformat used in Google Earth and related web services.

Tools For LBS Apps

Several companies provide developer-friendly tools and APIs as a value added service. Using these dramatically speeds up the development and deployment of location-aware services. Each tool normally focuses on one or a lesser range of mobile platforms.

Advertisement companies like Admob offer developers a stand-alone location aware advertisement program, to better target their offerings, while there are no map interfaces to be seen, just the coordinates.

Below are more links to maps and location based service resources:

- **Garmin Mobile XT SDK:** *developer.garmin.com*
- **Android offline maps project:** *code.google.com/p/big-planet-tracks/*
- **TeleAtlas:** *developerlink.teleatlas.com*
- **Nutiteq:** *www.nutiteq.com*
- **RIM:** *us.blackberry.com/developers/* (search for “map api”)
- **Nokia Maps:** *developer.here.net*
- **Nokia Windows 8 Mapexplorer:** *projects.developer.nokia.com/mapexplorer*
- **Windows Phone:** *msdn.microsoft.com/en-us/library/windowsphone/develop/ff431803*
- **Google Map resources:** *developers.google.com/maps/mobile-apps*

BY Shailen Sobeen & Richard Bloor

Near Field Communication (NFC)

Although there has been increased interest in Near Field Communications (NFC) technology during the last three years, NFC is actually not a new technology.

NFC is an evolution of Radio Frequency Identification (RFID) technology, which has been in operation since the early 90's[1]. NFC extends the capabilities of RFID and at the same time maintains backward compatibility with the older technology. For years, RFID has been used mainly for tracking objects and for simple tap-to-pay payment purposes. NFC technology, however, allows more than just payments to be made.

NFC is an interface and protocol built on top of RFID that is targeted at mobile devices. The technology operates on unlicensed ISM (Industry Scientific Medical) of 13.56 MHz which limits the range of operation to 3 cm or touch and provides the mobile devices with a means of secure communication without any network configuration. This is an added bonus over Bluetooth, which requires pairing for a communication to work.

1 www.nearfieldcommunication.org/history-nfc.htm

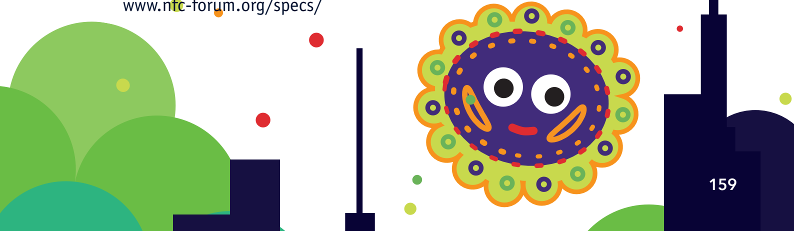
Advantage of NFC over Bluetooth[2]:

1. **Security**: NFC's short range of operation is actually an advantage. Some smartphone implementations require the screen to be active and that the PIN to be entered before access to NFC hardware is allowed.
2. **Low power consumption**: Due to the small range of operation, only a weak electromagnetic field has to be generated to write to or read NFC tags.
3. **Quick pairing**: NFC devices are able to connect within a tenth of a second.

Brief overview of NFC under the hood

NFC works by electromagnetic induction. When the electromagnetic waves in a coil changes, a voltage is induced. From a hardware perspective, a series of high and low voltages represent bits. This is how packets are sent between devices. In 2006, the NFC Forum, which oversees the NFC ecosystem, defined the NFC Specifications. The Forum established a standard for the NFC Data Exchange Format (NDEF), a light-weight binary message format that encapsulates data. For instance, to encode a URI they have a 5 byte header and can be as short as about 12 bytes to transmit a short URL. More information can be found online[3].

2 www.nearfieldcommunication.org/bluetooth.html
3 www.developer.nokia.com/Community/Wiki/Understanding_NFC_Data_Exchange_Format_(NDEF)_messages; www.nfc-forum.org/specs/



NFC Modes Of Operation[4]

1. **Reader/Writer mode:** In this mode, the smartphone acts as a reader. The smartphone generates electromagnetic fields to read NFC tags. It is only possible to write to an NDEF message on an NFC tag if the tag is not read-only.
2. **Peer-to-Peer mode:** This is a major extension to RFID technology. In P2P mode, two smartphones are able to exchange small amounts of information, such as vCards, URLs or initiate a Bluetooth connection for large data transfers. Android Beam typically functions in P2P mode where a "user-invisible" pairing takes place with NFC, and data transfer takes place over the faster Bluetooth connection.
3. **Card Emulation Mode:** Card emulation mode is, by far, the most interesting mode of operation. In this mode, the smartphone acts as a passive NFC tag. Card emulation mode is crucial for payment purposes in which sensitive information is stored on a secure element (more below). Google Wallet and Microsoft Wallet are examples of applications that allow an NFC-equipped smartphone to be used for tap-to-pay purposes and rely on card emulation. As of now, the latest Android version (Jelly Bean) does not have public APIs for Card Emulation mainly because there is not yet a standard for the NFC ecosystem. (See section Current Difficulties).

4 www.nfc.cc/technology/nfc/

The Secure Element

At the heart of any card-emulated device lies the secure element[5] that contains the secure data (credit card information amongst others[6]). As of now, there are three possible locations where the SE can be stored:

1. On the SIM/UICC (via Single Wire Protocol, a specification that allows a connection between the SIM card and the NFC chip.)
2. Inside the phone's NFC Chip
3. On an SD Card

As a side note, Google Wallet[7] stores only credentials of a Google prepaid credit card on the secure element of the phone's NFC chip. Only the Google credit card numbers are passed to merchants while actual credit card numbers are stored on secure Google servers. Microsoft Wallet on the other hand stores sensitive element on the secure element of the SIM card. According to Microsoft, such a method allows people to swap their wallets from one phone to another.

Around the world, a growing number of credit institutes and Mobile Network Operators (MNOs) are cooperating to deploy payment methods that use NFC. MasterCard PayPass and VISA PayWave are examples of such deployed solutions. NFC SIM cards are currently being issued by MNOs such as A1 (Austrian), France Orange and China mobile.

5 nearfieldcommunication.com/developers/nfc-architecture/
6 www.smartcardalliance.org/
7 www.google.com/wallet/faq.html

Current uses of NFC

1. Read/Writer mode: Basic RFID functionality involving an NFC tag. The tags can be used for numerous purposes (e.g defining profiles for a smartphone, hold some amount of data such as a URL or a contact information, amongst others).
2. P2P, involving two NFC-enabled smartphones: Pairing, Exchange of data.
3. Card Emulation: Ticketing, Payments, Switching operations (e.g opening a door), replacement of cards (health insurance cards, credit cards, driving license, amongst others).

Current difficulties

NFC is an exciting technology that will bring about more economic transactions. However, before we see a widespread deployment of payment methods using NFC, a full understanding and cooperation among all banks, hardware manufacturers, MNOs and operating system developers is necessary.

Also, some major phone manufacturers, have not yet adopted NFC technology. Current Apple devices, for instance, do not support NFC. The company has decided to rely on the Passbook application which has a completely different mode of operation compared to NFC-centric applications.

Furthermore, the secure element has a limited storage space. It is not clear how this space should be shared among all the players.

Lastly, because of the lack of accepted standards, some banks have deployed their own solutions and want to convince merchants to accept their new mode of payment.

Implementation of NFC

Windows Phone 8

The WP8.0 SDK provides the Proximity packages which provides a set of classes that provides the necessary APIs to enable P2P data sharing between WP8 applications. It is also possible to transfer small packets of data from an Android device to a WP8 device and vice-versa. As of now, the implementation is still in its infancy and it is not possible to transfer large amount of data. More implementation details can be found online[8].

Android

As from API Level 9 (Gingerbread 2.3), Android provided a set of high-level APIs that makes use seamlessly easy. More information can be found on the Android developer page[9].

Blackberry

The latest SDK also provides high level APIs for NFC purposes. Example code for implementing an NFC tag reader and writer[10] and further information on how to use card emulation mode on BlackBerry[11] can be found on RIM's websites.

8 msdn.microsoft.com/en-us/library/windowsphone/develop/jj207060

9 developer.android.com/reference/android/nfc/package-summary.html

10 docs.blackberry.com/en/developers/deliverables/34480/Near_Field_Communication_1631111_11.jsp

11 http://supportforums.blackberry.com/t5/Java-Development/NFC-Card-Emulation-Primer/ta-p/1596893

BY Bob Heubel

Implementing Haptic Vibration

Design Considerations

Why should you use haptic vibration effects to your app? Your app will run just as well without the tactile feedback, right? Yes, possibly, but you will lose the one sensory element that makes your virtual environment more realistic and compelling. Margaret Atwood once wrote, “Touch comes before sight, before speech. It is the first language and the last, and it always tells the truth.” It is this sense of touch feedback, more than sight or hearing that teaches us what to expect from interactions in the real world.

It is our experiences in the real world that define a user's expectiations in the virtual world found in your apps.

Take a button press as an example. A real button press is a very tactile experience. It has a beginning and an end. You feel a satisfying confirmation of your action. In comparision, a virtual button press feels hollow without a Haptic effect to simulate that same confirmation of action. More than this, without a Haptic tactile confirmation you force the user to rely on visual/audio cues that are more stressful to process than simply using our sense of touch.

Haptic feedback is even more important in mobile video games. We know this from our experience with console games. Remember when the Sony PS3 launched without "DualShock" rumble pad motors? Gamers voiced their dissatisfaction and shortly after Sony brought the DualShock rumble feedback to the PS3. The same Haptic feedback satisfaction applies to mobile games. Using Haptic effects in your games will help to give your mobile users what they already expect from console platforms. And if you design well, your games will feel more realistic and compelling to your users.

When designing a Haptic experience, keep in mind the ultimate experience of the user. Spend some time planning before starting your Haptic implementation. Once the project is defined and taking shape in your mind, consider the following guidelines:

- Simple sensations are often the most effective. It is sometimes surprising to realize that something like a very simple Pop or Click sensation can enhance menu interactions and increase user confidence within the application.
- Sensations synchronized with audio and visual events, like a simple button click event, make the whole greater than the sum of its parts. Seeing, hearing, AND feeling an object or activity promotes sensory harmony in a way just seeing and hearing alone cannot.
- It is bad to annoy the user. Poorly chosen or designed touch sensations can be annoying and counterproductive. While a high-pitched buzz may be very effective as part of an alert, continuous reoccurring buzzing will eventually cause a user to leave an application annoyed.
- It is bad to confuse and overwhelm the user. Just as too many beautiful sounds played simultaneously become a cacophony, too many compelling touch sensations played together or too close to each other in time and space can become confusing and overwhelming.
- Familiarity eases the user experience. Haptic effects can relay important information to a user, which might not be available or practical to provide through graphics or sound. Standardization and consistency are important. Limiting

the Haptic effect language to a manageable, reused set of sensations makes the user's learning process easier because there are fewer Haptic effects to recognize.

Nearly all mobile platforms allow for some form of haptic vibration feedback control. This section will be your resource for understanding the classes and methods between these platforms.

iOS

iOS may have the least amount of vibration documentation for developers as Apple currently gives developers little vibration control for their devices. The iOS vibration method below applies to iPhones only. iPads and iPods currently do not support vibration.

Use the `SysSoundViewController` Class[1] with the `AudioServicesPlaySystemSound` function and the `kSystemSoundID_Vibrate` constant to trigger vibration on your iPhone device. Calling this constant will turn your motor on for a set duration of about 2 seconds.

Android

Android is unique for vibration control. It provides native support and has more vibration control than iOS. Furthermore, there are ways to extend this Android vibration control for developers so they can create more console-like X-Box or PlayStation feedback experiences. But whether you use the basic or extended methods below, please note that a user may have enabled haptic effects for better accessibility. For instance the KickBack Accessibility Service provides haptic feedback and

1 developer.apple.com/search/index.php?q=SysSoundViewController

is available as part of the eyes-free[2] open source project. So, consider how haptic effects generated by your application may interact with, or disturb, such services.

For basic vibration control in Android, you must first grant permission `android.permission.VIBRATE` to allow your application to vibrate. Next you use the `Vibrator`[3] Class with `getSystemService` function and the `Context.Vibrator_Service` to call the vibration service.

Within the above method you can vary the duration of the vibration event in milliseconds and set vibration patterns by setting up as many of start and sleep events as you like. The basic Android vibrate control method only lets you control the duration of vibration events.

Extended Android Vibration Control

Because the Android platform is open source, there is at least one company that offers free methods to extend Android's vibration control. Immersion Corporation's Haptic SDK[4] allows full vibration motor control of duration, amplitude and pulsing frequency with a library of over 120 pre-defined Haptic vibration feedback effects. With this type of control, application developers have the capability of designing vibration effects rivaling console gaming vibration experiences while also conserving battery power.

For Android developers using Unity3D Pro, Immersion offers this same extended method through a plug-in, also found on their main SDK webpage. Developers interested in this extended vibration control can download the company's Quick Start Guide[5] that explains how to set-up your Eclipse environment

2 code.google.com/p/eyes-free
3 developer.android.com/index.html#q=Vibrator
4 immersion.com/haptic/sdk
5 immersion.com/haptic/guide

and use the `Launcher` method to call Haptic effects from the pre-defined library. On Google Play you can also download a showcase app and feel the pre-designed Haptic effects before using them in code. It is called the "Haptic Effect Preview" app.

Besides having pre-designed Haptic effects for developers to use there is a hardware abstraction layer in the Immersion library that compensates for the differences in motor types between hardware manufacturers.

BlackBerry 10

BlackBerry gives you the same basic on/off vibration control that Android does, but without an extended method. For BlackBerry you use the `VibrationController`[6] Class with `startVibrate(int duration)` and `stopVibrate(int duration)`

In addition, Blackberry now has an `intensity` (1-100) parameter for developers to play with.

Windows 8

Windows offers a basic method for vibration control, but no extended method at this time. Use the `VibrateController`[7] Class with `Start` & `Stop` Methods to vibrate your device motor from 0-5 seconds. For finer duration control you will need to set a `TimeSpan` method in order to use millisecond values.

The Windows 8 class listed above is the same as the previous Windows 7 class.

6 developer.blackberry.com/search/?search=VibrationController
7 social.msdn.microsoft.com/search/en-us?query=VibrateController

BY Mostafa Akbari

Implementing Augmented Reality

Augmented reality is a technology that enhances the real world with virtual elements. Real and virtual components are combined in real time to augment the perception of reality by adding additional information to a real world environment. Although visual augmented reality is the most common form, this technology can appeal to all our senses. Added information could include: a three-dimensional object, which blends with the actual surroundings; a two-dimensional overlay containing text; or simply an audio file.

The prognosis for 2013 is 200 million augmented reality users. By 2020 the global number of consumers who use AR applications, is supposed climb up to one billion. The growth of the AR market is increasing exponentially, according to ‘Research and Markets’. With a sales growth of 181 million US$ in 2011 an increment at a growth rate of 95.35% up to 5,155 million US$ until 2016 is to be expected.[1]

AR Usage Scenarios in Mobile

Mobile AR is used in situations where additional information can increase the efficiency, effectivity and joy of use while on the move. Mobile AR is especially suitable for those applications where people are confronted with lots of data or a heavy information load and need to process it in a short time period. Through the insertion of virtual information into the live stream on the screen, the user's attention no longer needs to switch between the screen of the mobile device and the

1 All numbers from www.researchandmarkets.com/reports/1963197/

environment. Mobile Augmented Reality solutions have plenty application areas, such as in the industry, marketing, education or just for entertainment. Here are just some examples.

- **AR Jump n' Run:** AR Jump n' Run is a location based application for Android smartphones developed with DroidAR[2]. The game can be played both indoors and outdoors, by either using the global positioning system (GPS), step detection, or both. The player walks through a virtually-enriched world by using his Android device as the viewport and collects 3D items. These items might be hostile or friendly and have different consequences for the gameplay. The game also features an in-game map editor that allows players to create and modify maps directly on their devices.
- **Augmented Reality Browsers:** AR-browsers such as Layar and Wikitude can superimpose the live view of the physical, real-world environment around you with location-based data. The location of the user is determined by GPS and information about POIs (points of interest) nearby is displayed on the screen of the smartphone. Mostly the information is displayed as clickable icons or text fragments. Wikitude additionally offers a connection to Wikipedia for more information. Download and try the apps at the Wikitude app website[3] and Layar's website[4].
- **IKEA Catalogue App:** The IKEA AR App is build by and with Metaio[5]. Instead of using QR codes the app relies on image recognition software from Metaio. By scanning Ikea's catalogue pages with a special AR "Unlock"-Logo

2 code.google.com/p/droidar/
3 www.wikitude.com/app
4 www.layar.com/features/#feature-layarapp
5 www.metaio.com

additional information, like customization options and further item pictures are displayed. The user can also see how furniture from IKEA will look in his home environment by placing 3D models in the device's camera picture.

More information on the feature set and a video presentation can be found on Metaio's developer blog[6].

Tracking

Location determination is an essential part of modern augmented reality systems. It is necessary to determinate the position of the user and to place virtual objects correctly into a three-dimensional room. Common tracking technologies include GPS, optical sensors, compass, accelerometer, gyroscope and step detection. Another important concept is the use of marker-based and markerless tracking.

Marker-based Tracking

Markers are a simple and inexpensive solution to identify objects. Furthermore, the accuracy is very high, as long as the distance between the marker and the camera is not too far. Marker tracking operates with the principle of pattern recognition. The markers are often black and white, and square to facilitate the acquisition and processing of orientation done by an image processing system. This processing system is able to operate in the range of normal light or infrared light. Processing the information about the actual size of the marker, the distance between the device and the marker can be calculated. It is also possible to use spherical reflectors which reflect the incident light in the direction from which it came.

6 augmentedblog.wordpress.com/2012/07/24/ikea-2013-catalog-has-augmented-reality/

Markerless tracking

A similar but slightly different implementation scenario uses natural features instead of markers. These features can be two-dimensional patterns (e.g. advertisement posters) or even three-dimensional surroundings (e.g. buildings). The recorded images are compared with a database to detect a match, which requires complex algorithms and high processing power. Changes in lighting conditions and movement of the object can make it difficult to find the right match. Another challenge is the recognition of objects whose shape is not static.

Hybrid Tracking

Hybrid tracking technology combines the different sources of position data, such as GPS, 3D feature detection, marker detection and step detection. This allows a higher positioning and motion detection accuracy.

Augmented Reality SDKs

- **Wikitude SDK:** Wikitude is a location-based Point of Interest (POI) Browser. A classic usage scenario where Wikitude offers an adequate solution is POI search ("Where is the nearest post office?"). Wikitude is designed for static content and does not allow interactive scenarios. If you use newer versions to build your own Wikitude browser app, you can also use HTML5, Titanium or Phonegap. Even BlackBerry 10 is supported.
  www.wikitude.com/developer
- **Vuforia:** The Vuforia SDK from Qualcomm only supports 2D feature extraction and only recognizes locally stored special images. Vuforia cannot be used to create location based applications which use movement data or geo-references.
  www.vuforia.com

- **Metaio:** The Metaio SDK is one of the most advanced AR development tool sets and has its main focus on the augmentation of 2D images e.g. in magazine or catalogs (like the Ikea app mentioned before). The pro version of the SDK also supports the recognition of 3D objects like a product package, a statue or the facade of a building. As a downside, the SDK offers very limited possibilities to implement individual interaction scenarios. The developer can only decide between 2D or 3D feature extraction and location based objects.
  www.metaio.com
- **Layar:** In the beginning Layar was a pure location-based AR platform with layers to fade in and with very limited interaction possibilities for the user. Now, Layar changed its focus on 2D feature extraction to augment images. The new SDK and related tools (called Layer creator) are specially designed for extending print media content. The Layar Player SDK for iOS makes it possible build Layar Apps which do not need the Layar browser.
  http://www.layar.com
- **DroidAR:** DroidAR SDK is built for the development of interactive location-based AR applications. DroidAR provides hybrid tracking and marker based AR but no augmentation of 2D images. Its strength lies in building apps which are used while on the move, e.g. AR games or shopping applications. At the moment DroidAR supports only Android devices.
  http://code.google.com/p/droidar/

— **D’Fusion:** The D’Fusion SDK by Total Immersion offers 2D feature extraction to augment images and is very similar to other SDKs like the one from Metaio, Qualcomm and Layar. Within some SDK bundles you get face tracking and movement detection libraries.
www.t-immersion.com

— **ARToolwork:** ARToolworks offers SDKs such as NyARToolKit. The SDKs offer only marker-based augmented reality and are distributed under a fair licence model.
www.artoolworks.com/products/mobile/

SDK feature set	Wikitude SDK	Vuforia Qualcomm	Metaio	Layar	DroidAR	D’Fusion Total Immersion	ARToolworks
Location-based AR	🟩	🟥	🟩	🟩	🟩	🟥	🟥
iOS and Android	🟩	🟩	🟩	🟩	🤖	🟩	🟩
SDK for interactive AR	🟥	🟩	🟩	🟥	🟩	🟩	🟥
Hybrid Tracking	🟥	🟥	🟥	🟥	🟩	🟥	🟥
Marker Detection	🟥	🟥	🟥	🟥	🟩	🟥	🟩
2D Picture Augmentation	🟩	🟩	🟩	🟩	🟥	🟩	🟥
Scale potential	🟩	🟩	🟩	🟩	🟩	🟩	🟥
Step Detection	🟥	🟥	🟥	🟥	🟩	🟥	🟥
Developer Forum	🟩	🟩	🟩	🟩	🟩	🟩	🟩
License costs	🟩	🟥	🟩	🟩	🟥	🟩	🟩

AR Developing 101

This section will give you a general introduction to the key concepts needed to create AR applications. Once you understand these concepts, you should be able to chose the right framework for your project.

The Real And The Virtual World

AR means placing artificial objects into the real world somehow, using a **virtual layer**. This virtual layer, or virtual world, and its coordinate system is tied to the real world by reference points. This reference can be a GPS position in case of a tourist guide app that provides location-based information about POIs. It can also be a visual marker, for example a game printed on a cornflakes box, which could be played anywhere in the world and only needs this relative reference to the cornflakes box.

Mapping Between The Two Worlds

For location-based AR apps a mutual mapping between the constantly changing position in the real world and the position in the virtual world is needed. Rendering engines like OpenGL v1 and v2 reduce the complexity of this process and increase speed and real-time accuracy. The engine also takes care of matching the camera's virtual and real-world position and guarantees fluent camera transitions when these positions change. To make your life easier as a developer you can use extended engines like gameplay[7] or Unity[8].

As the central element, the camera data can also be passed to other components. For example, a collision component can easily calculate the distance between virtual objects and the image captured by the camera.

7 www.gameplay3d.org/

8 unity3d.com/

Creating Virtual Elements

Virtual items are represented as 3D or 2D objects which might have different behaviour logics: Some objects might be collectable, others follow the user, or they may remain static and allow no interaction at all.

To create this kind of virtual elements you need a technique called Simultaneous Localization And Mapping (SLAM). This is a very powerful Computer Vision technique based on uniquely identifiable sections in an image called features. These features are merged with the real-world in a 3D space in which the device can move around. The created 3D cloud can also be used for real-world object or shape detection and then augment these. Another part of Computer Vision is image recognition. The recognized image position can be used as the reference position for the virtual world (compare to the example with the cornflakes box). SLAM and Computer Vision technique require immense computational powers to run in real time. The limitations and the capabilities of the hardware have to be taken into account.

Combining Application Layers

An important element of visual AR is to draw something over the image received from the camera. Depending on your app requirements, you will want to place 2D or 3D graphics in this overlay and use the respective APIs.

A 2D overlay is usually sufficient for simple POI browsers. We strongly recommend a rendering framework like OpenGL is used, rather than re-inventing the wheel. That framework will use the user's position, device orientation, other sensor information, or the image analysis data and translate it for displaying your content accordingly. If the rendering component is decoupled from the rest of the app's code it can be exchanged in future, for instance to switch to more advanced rendering solutions.

Maybe you want to add things like a small radar UI to visualize the position of the virtual objects or some simple buttons to interact with the virtual world. Then make sure to stick to platform-specific patterns and designs. And implement these elements on a separate layer to remain flexible.

Composing A Virtual World With Multiple Layers

The best practice for composing the virtual world is to use a tree structure and place virtual objects in different layers: One layer for fixed "background" objects which do not need to be updated or which do not interact with the user and other layers for movable objects or UI elements.

You should only update and render objects close to the user and make use of the quad tree structure. A quad tree is a data-structure which allows to obtain all objects in a bounding box in an efficient way. Depending on the device hardware a different view radius can be used to keep the application performant.

We also recommend to use an update mechanism which triggers the updates and calls update method e.g. every 20 ms. The nodes in the object tree individually decide to which children these update calls should be forwarded. A quad tree for example will only update the objects close to the user to keep the update procedure efficient. A basic list structure would update all its children and is more suitable for elements which do not have a virtual position (like logical game stats) or which have to be updated at all. The exact object composition concept depends on the application scenario and cannot be defined in an universal way.

BY Dean Churchill

Application Security

Readers of this guide know how widespread smart mobile devices have become and how useful mobile apps can be. We use these powerful, connected and mobile computers for a multitude of things every day. 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'.

Companies are now developing apps that 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).

Clearly mobile apps are going to attract the attention of hackers and thieves whose interests extend well beyond getting a 99 cent app for free. The historical network and endpoint based defenses (like anti-virus tools) are not enough. Embedding security into the mobile application is critical.

The architecture of mobile apps continues to evolve. Some apps are native-only, and require distinctly different code bases for each different mobile operating system. Some are web-views, little more than a web site url wrapped in an icon. Others are hybrids, a combination of native app functionality with web views. Most mobile apps need to connect with backend services using web technologies to fetch or update information. Like web apps, classic application security needs to be used with mobile apps. Input needs to be validated for size, type, and values allowed. Error handling needs to provide useful error messages to users that do not leak sensitive information. Do not print stack traces or system diagnostics that hackers can leverage to penetrate further. Penetration testing of applications is needed to assure that identification, authentication and authorization controls cannot be by passed. 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 should 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 applications can covertly make phone calls and send SMS messages to expensive numbers.

The application developers may be concerned about their 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 (or malware) that could steal your users' data 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 to 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[1] correctly is an easy way to make it harder to understand the code.

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

Hiding Control-Flow

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

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

1 gcc.gnu.org/wiki/Visibility

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

Protecting Network Communications

Network communications are also vulnerable, particularly when apps can be installed in emulators or simulators, where network analyzers are freely available and able to monitor and intercept network traffic. Consider protecting sensitive network communications, for instance by using SSL for HTTP traffic between your app and servers. Even then MATE attacks, especially over WiFi networks, may disclose sensitive data.

Protect Against Tampering

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

Communications can be monitored and hacked between the mobile app and backend services. Even when using SSL, an intercepting web proxy (like Paros) can be setup on a WiFi connection that will inspect SSL 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 data in the mobile application and on the server, so that network sniffers will only ever see encrypted data.

Protecting Cryptographic Algorithms

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

Best Practices

Do Not Store Secrets or Private Info

Minimize the amount of sensitive information stored on the device. Do not store credentials or encryption keys, unless secure storage is used protected by a complex password. Instead, store authentication tokens that have limited lifetime and functionality.

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

Do Not 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 – most OS protections can be bypassed trivially by jailbreaking the device. Do not trust that native keystores will keep data secret – keystores can be broken by bruteforce guessing unless the user protects the device with a long complex password.

Minimize Permissions

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

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

Tools

Protection

Basic Java code renaming can be done using Proguard[2], an open-source tool and Arxan's GuardIT[3].

Two vendors for managed-code (Java and .NET) protection tools are Arxan Technologies[4] and PreEmptive Solutions[5].

The main vendors for native code protection tools and white-box cryptography libraries are Arxan and Irdeto[6].

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[7]. A standard network sniffing tool available on common platform is Wireshark[8].

2 www.proguard.sourceforge.net
3 arxan.com
4 arxan.com
5 preemptive.com
6 www.irdeto.com
7 sourceforge.net/projects/paros
8 sourceforge.net/projects/wireshark

De-Compiling

See the Hex Rays de-compiler[9].

Learn More

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

- Apple provides a general guide to software security[10]. It also includes several links to more detailed topics for their platform.
- Commercial training courses are available for iOS and Android[11], and Lancelot Institute[12] provide secure coding courses covering iOS and Android.
- O'Reilly (2011) published a book on Android security Jeff Six: Application Security For The Android Platform. Processes, Permissions and Other Safeguards (Dec 2011)[13] and another for iOS, Jonathan Zdziarski: Hacking and Securing iOS Applications[14] .
- Charlie Miller et al. (2012) published iOS Hackers Handbook[15], which demonstrates how easy it is to steal code and data from iOS devices.

9 www.hex-rays.com
10 developer.apple.com/library/mac/navigation/#section=Topics&topic=Security
11 marakana.com/training/android/android_security.html
12 www.lancelotinstitute.com
13 shop.oreilly.com/product/0636920022596.do
14 shop.oreilly.com/product/0636920023234.do
15 www.wiley.com/WileyCDA/WileyTitle/productCd-1118204123.html

- Academic researchers demonstrate how much information can be gleaned from public Android apps at USENIX 2011[16].
- A free SSL tester is provided by Qualsys Labs[17].
- Extensive free application security guidance and testing tools are provided by OWASP[18], including the OWASP Mobile Security Project[19].
- An open-source mobile application performance monitoring tool for Android is provided by AT&T's Application Resource Optimization tool[20].

The Bottom Line

Mobile apps are becoming ever more trusted, but they are exposed to many who would like to take advantage of that trust. The appropriate level of application security is something that needs to be considered for every app. In the end, your app will be in-the-wild on its own and will need to defend itself against hackers and other malicious threats, wherever it goes.

Invest the time to learn about the security features and capabilities of the mobile platforms you want to target. Use techniques such as threat modelling to identify potential threats relevant to your application. Perform code reviews and strip out non-essential logging and debugging methods. Consider how a hacker would analyze your code, then use similar techniques, in a safe and secure environment, against your application to discover vulnerabilities and mitigate these vulnerabilities before releasing your application.

16 static.usenix.org/event/sec11/tech/slides/enck.pdf
17 https://www.ssllabs.com/ssltest/
18 www.owasp.org
19 https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
20 developer.att.com

Testing Your Application

After all your hard work creating your application how about testing it before unleashing it on the world? Testing mobile applications used to be almost entirely manual, thankfully automated testing is now viable for many of the mobile platforms. Several of the major mobile development platforms include test automation in the core tools, including Android and iOS.

Cross-platform test automation tools are available for popular platforms; some are free-of-charge and open-source, others are commercial.

This chapter covers the general topics; testing for specific platforms is covered in the relevant chapter. We will start by covering several key concepts which need to be considered before the code is written as they affect how it will be written. The topics include:

— Testability
— Test-Driven Development (TDD)
— Unit testing

Testability: The Biggest Single Win

If you want to find ways to test your application effectively and efficiently then start designing and implementing ways to test it; 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 tests require less maintenance.

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

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

Test-Driven Development

There are several ways to design and implement software. Test-Driven Development (TDD) is an approach where developers write automated tests in parallel with writing the main code for the app. The automated tests will include unit-tests, these are covered in the next topic.

TDD is both a mindset and a practice. It requires a certain amount of discipline to write the tests even when the going gets tough. And by practising TDD diligently developers are likely to write better-quality, simpler, cleaner code which is also easier to maintain in future (as they are protected and supported by a set of automated tests which can be run when maintaining and revising the source code of the app).

The pure approach is when the tests are written first, and run, before new application code is written. The new tests are expected to fail, that is they should report failures in the behaviour of the app. The failures express the mismatch between what the app needs to do and what it currently does. Now the developer has a simple, automated way to test their modifications to the source code for app. Once just enough software has been written to get all the tests to pass we now have confidence the app meets the requirements specified by these tests.

While we may have met the business requirements we may decide our work is not 'done' yet. For instance there may be duplication, unnecessary complexity, and other known flaws in the implementation. We now have an opportunity to revise and improve the source code through a process known as 'refactoring'. Refactoring is where developers improve the implementation where the automated tests continue to pass when run against the improved code.

Sometimes we need to modify or even remove existing automated tests when the desired changes 'break' the existing code. For rapidly changing code bases the extra work of working on the tests can be perceived as an unnecessary burden. Teams need to decide and commit to revise their automated tests at some point before they finish their changes; otherwise many of the long term benefits of these tests would be lost.

TDD has become more popular and widespread in the general development communities, particularly when using Agile Development practices.

Although TDD is a struggle when using the current Mobile Test Automation tools several people have provided examples of using TDD successfully, for instance Graham Lee's book Test-Driven iOS Development[1]. You can also consider using TDD for the generic aspects of the app.

Unit Testing

Unit testing involves writing automated tests that test small chunks of code, typically only a few lines of source code. The source code may be the implementation of an individual 'method' or function for instance that checks whether an

1 www.informit.com/store/product.aspx?isbn=0321774183

item of data is correctly formatted, or not. Unit tests have a long pedigree in software development, where JUnit[2] has spawned similar frameworks for virtually all of the programming languages used to develop mobile apps.

Unit tests can be written by anyone who can write software, however to get maximum benefit generally they should be written by the same developer who writes the source code for the app.

Unit tests are only one aspect of automated testing, they are not sufficient to **prove** the app works. They help developers to understand what individual pieces of the software is expected to do. Additional testing, including other forms of automated tests can help to increase our confidence in the app.

Testing Through The Five Phases of an App's Lifecycle

The complete lifecycle of a mobile app fits into 5 phases: implementation, verification, launch, engagement and validation. Testing applies to each phase. Some of the decisions made for earlier stages can affect our testing in later stages. For instance, if we decide we want automated system tests in the first phase they will be easier to implement in subsequent phases.

2 en.wikipedia.org/wiki/JUnit

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 our implementation is done well.

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

- Do we use test-driven development (TDD)?
- Do we write unit tests even if we are not using TDD?
- Will we have automated system tests? If so, how will we facilitate these automated system tests? For instance by adding suitable labels to key objects in the UI.
- How will we validate our apps? For instance, through the use of Mobile Analytics? Crash reporting? Feedback from users?

Question the design. We want to make sure it fulfills the intended purposes; we also want to avoid making serious mistakes. Phillip Armour's paper on five orders of ignorance[3] is a great resource to help structure your approach.

Also consider how to improve the testability of your app at this stage so you can make your app easier to test effectively and efficiently. Practices, including unit tests and Test-Driven-Development (TDD) apply to the implementation phase. Remember to test your build process and build scripts to ensure they are effective, reliable and efficient, otherwise you are likely to suffer the effects of poor builds throughout the life of the app.

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

Phase 2: Verification

This includes reviewing unit tests, internal installation, and system tests.

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

For apps that need installing we need ways to deploy them to specific devices for pre-release testing. For some platforms (including Android, iOS and Windows Phone) the phones need to be configured so the apps can be installed. We also need to decide which phones to test the app on. For instance, it is wise to test the app on each suitable version of the mobile platform. For iOS this may only include the latest releases. On Android it already looks pretty different because low end devices are still being sold with version 2.x of Android and will never be updated to 4.x

We will also want to test different form-factors of devices; for instance where the ratio of the screen dimensions differ. The iPhone 5's new screen dimensions exposed lots of UI bugs. Android developers are well aware of the many issues different screen sizes can trigger.

System tests are often performed interactively, by testers. Consider evaluating test automation tools and frameworks for some of your system tests. We will go into more detail later in this section.

We also want to consider how we will know the app meets:

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

Phase 3: Launch

This includes pre-publication and publication.

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

Given these constraints it is worth extending your testing to include pre-publication checks of the app such as whether it is suitable for the set of targeted devices. The providers of the main platforms now publish guidelines to help you test your app will meet their submission criteria. These guidelines may help you even if you target other app stores.

Apple	https://developer.apple.com/appstore/resources/approval/guidelines.html (Apple account needed for access)
Android	http://developer.android.com/distribute/googleplay/publish/preparing.html#core-app-quality
Windows Phone	http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh394032(v=vs.105).aspx
BlackBerry	http://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. We can test each aspect of this phase. Try searching for your app on the relevant app store, and in mainstream search engines. How many different ways can it be found by your target users? What about users outside the target groups - do you want them to find it? How will users trust your app sufficiently to download and try it? Does your app really need so many permissions? How large is the download, and how practical is it to download over the mobile network? Will it fit on the user's phone, particularly if there is little free storage available on their device? And does the app install correctly - there may be signing issues which cause the app to be rejected by some phones.

Phase 5: Validation

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

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

- Problem detection and reporting. These may include our 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?

Interactive Testing

Variety and movement can help expose bugs which remain dormant when testing on a small set of devices in a fixed location such as your workplace. Learn from your users - how do they use your app, or similar apps? Then devise tests that mimic the ways they use apps and devices.

The guidelines at *appqualityalliance.org/resources* are worth considering when devising your test cases. For instance they include testing the app to see what happens when an incoming phone call is received; and when the user switches the phone to 'flight mode'.

The next few sections will describe three different approaches to interactive testing.

— **Physical devices:** why testing with real phones is important.
— **Remote control:** a way you can test using phones that are not physically in your hands, where they may be thousands of miles away and even on another continent.
— **Crowd sourced testing:** where other testers perform testing on your behalf.

Physical Devices

Although emulators and simulators can provide rough-and-ready testing of your applications, and even allow tests to be fully automated in some cases, ultimately your software needs to run on real phones, as used by your intended users. The performance characteristics of various phone models vary tremendously from each other and from the virtual device on your computer. So you should buy, beg, borrow various 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 we 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.

Remote Control

If you do not have physical devices at hand or if you need to test your application on other networks, especially abroad and for other locales, then one of the 'remote device services' might help you. They can help extend the breadth and depth of your testing at little or no cost.

Several manufacturers provide this service free-of-charge for a subset of their phone models to registered software developers. Both Nokia[4] (for MeeGo and Symbian) and Samsung[5] (for Android and Bada) provide restricted but free daily access.

You can also use commercial services of companies such as PerfectoMobile[6] or DeviceAnywhere[7]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.

Crowd-Sourced Testing

There are billions of users with mobile phones across the world.

Some of them are professional software testers, and of these, some work for professional out-sourced testing service companies such as uTest and mob4hire. They can test your application quickly and relatively inexpensively, compared to maintaining a larger dedicated software testing team.

These services can augment your other testing, we do not recommend using them as your only formal testing. To get good

4 apu.ndhub.net/devices
5 rtl.innovator.samsungmobile.com/
6 www.perfectomobile.com
7 www.deviceanywhere.com

results you will need to devote some of your time and effort to defining the tests you want them to run, and to working with the company to review the results, et cetera.

Test Automation

Automated tests can help you maintain and improve your velocity, your speed of delivering features, by providing early feedback of problems. To do so, they need to be well-designed and implemented. Otherwise you risk doubling your workload to maintain a mess of broken and unreliable automated tests as well as a broken and an unreliable app. Good automated tests mimic good software development practices, for instance using design patterns[8], modularity, performing code reviews, et cetera.

It is important to assess the longevity and vitality of the test automation tools you plan to use, otherwise you may be saddled with unsupported test automation code. Test automation tools provided as part of the development SDK are worth considering. They are generally free, inherently available for the particular platform, and are supported by massive companies.

BDD Test Automation

BDD stands for Behavior-Driven Development[9] where the behavior is described in formatted text files that can be run as automated tests. The format of the tests are intended to be readable and understandable by anyone involved with the software project. They can be written in virtually any human language, for instance Japanese[10], and they use a consistent,

8 en.wikipedia.org/wiki/Design_Patterns Design Patterns
9 en.wikipedia.org/wiki/Behavior-driven_development
10 github.com/cucumber/cucumber/tree/master/examples/i18n/ja

simple structure with statements such as **Given, When, Then** to structure the test scripts.

There are various BDD frameworks available to test mobile apps. These include:

- **Calabash for Android and iOS:** *http://github.com/calabash*
- **Frank for iOS:** *www.testingwithfrank.com*
- **RoboGerk for Android:** *http://github.com/leandog/RoboGherk*
- **Zucchini for iOS:** *www.zucchiniframework.org*

and various implementations that integrate with Selenium-WebDriver for testing web apps, including web apps on iOS and Android.

Often, custom 'step-definitions' (small scripts that interact with the app being tested) need to be written by someone with coding skills.

GUI Test Automation

GUI test automation is where automated tests interact with the app via the Graphical User Interface (GUI). It is one of the elixirs of the testing industry, many have tried but few have succeeded in creating useful and viable GUI test automation for mobile applications. One of the main reasons why GUI test automation is so challenging is that the User Interface is subject to significant changes which may break the way automated tests interact with the app.

For the tests to be effective in the longer term, and as the app changes, developers need to design, implement and support the labels and other hooks used by the automated GUI tests. Both Apple, with UI Automation[11], and more recently Android[12] use the Accessibility label assigned to UI elements as the de-facto interface for UI automation.

2012 has seen lots of new contenders of test automation tools and services. Some commercial companies have open-sourced their tools GorillaLogic's MonkeyTalk[13] and LessPainful's Calabash[14]. These tools aim to provide cross-platform support particularly for Android and iOS. The companies charge for consulting and other services, the software is free to use.

Sadly several opensource projects appear to be mothballed including several we mentioned in earlier editions. Others including Robotium[15] and Frank[16] are doing well and may even have been incorporated into other test automation tools.

Headless Client

The user-interface (UI) of a modern mobile application can constitute over 50% of the entire codebase. If your testing is limited to using the GUI designed for users you may needlessly complicate your testing and debugging efforts. One approach is to create a very basic UI that is a thin wrapper around the rest of the core code (typically this includes the networking and business layers). This 'headless' client may help you to quickly isolate and identify bugs e.g. related to the device, carrier, and other environmental issues.

11 developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/UsingtheAutomationInstrument/

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

13 www.gorillalogic.com/testing-tools/monkeytalk

14 https://github.com/calabash

15 code.google.com/p/robotium/

16 testingwithfrank.com/

Another benefit of creating a headless client is that it may be simpler to automate some of the testing e.g. to exercise all the key business functions and/or to automate the capture and reporting of test results.

You can also consider creating skeletal programs that 'probe' for essential features and capabilities across a range of phone models e.g. for a J2ME application to test the File Handling where the user may be prompted (many times) for permission to allow file IO operations. Given the fragmentation and quirks of mature platforms such probes can quickly repay the investment you make to create and run them.

Beware Of Specifics

Platforms, networks, devices, and even firmware, are all specific. Any could cause problems for your applications. Test these manually first, provided you have the time and budget to get fast and early feedback.

Many mobile applications include algorithms, et cetera, unrelated to mobile technology. This generic code should be separated from the platform-specific code. For example, on Android or J2ME the business logic can generally be coded as standard Java, then you can write, and run, automated unit tests in your standard IDE using JUnit. Consider platform-specific test automation once the generic code has good automated tests.

$$$
monetize

Monetization

Finally you have finished your app or mobile website and polished it as a result of beta testing feedback. Assuming you are not developing as a hobby, for branding exposure, et cetera, now it is time to make some money. But how do you do that, what are your options?

In general, you have the following monetization options:

1. **Pay per download:** Sell your app per download
2. **In-app payment:** Add payment options into your app
3. **Mobile advertising:** Earn money from advertising
4. **Revenue sharing:** Earn revenue from operator services originating in your app
5. **Indirect sales:** Affiliates, data reporting and physical goods among others
6. **Component marketplace:** Sell components or a white-label version of your app to other developers

When you come to planning your own development, determining the monetization business model should be one of the key elements of your early design as it might affect the functional and technical behavior of the app.

Pay Per Download

Using pay per download (PPD) your app is sold once to each user as they download and install it on their phone. Payment can be handled by an app store, mobile operator, or you can setup a mechanism yourself.

When your app is distributed in an app store – in most cases it will be one offered by the target platform's owner, such as Apple, Google, RIM, Microsoft or Nokia – 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.

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. Premium SMS is still very popular for mobile web applications, Java games, wallpaper and ringtones.

Other operator APIs enable you to include features such as MMS, Call Back and Multimedia Conference in your app and earn revenue from their use. However, operator billing has been quite difficult to handle particularly if you want to sell in several countries, as you needed to sign contracts with each operator in each country.

In 2011, 57 of the world's largest mobile phone network operators and manufacturers founded the Wholesale Applications Community (WAC)[1], a non-profit organization that helps to standardize the mobile applications ecosystem. One of their key products is the WAC OneAPI, allowing a developer to easily interface with all connected operators. As recently disclosed[2], WAC is now integrated into GSMA. OneAPI provides a server based REST interface for operator billing and premium SMS.

Each operator will take a revenue share typically 45% to 65% of the sale price, but some operators can take up to 95% of the sale price (and, if you use them, a mediator will take its share too). Security (how you prevent the copying of your app)

1 wacapps.net

2 oneapi.gsma.com/developer-article-oneapi-and-the-wholesale-applications-community-wac/

and manageability are common issues with PPD but for some devices this might be the only option.

As of Android 4, Google decided to ask for Credit Card data at sign-up, something that Apple already required since 2008, which according to analysts is the key differentiator for higher monthly per app revenue. In addition to customers on monthly billing arrangements, pre-pay customers can use their pre-paid credits to purchase apps. Like BlueVia’s in-app payment API, this is particularly significant for developers targeting emerging markets where credit cards ownership is low.

It is worth noting that most of the vendor app stores are pursuing operator billing agreements, with Nokia Store having by far the best coverage with operator billing available in 46 countries. In addition, Nokia will be bringing its expertise to Microsoft's Windows Phone Marketplace to rapidly expand its operator billing coverage. Google and RIM are actively recruiting operators too. 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, at least, also insulates developers from the variation in operator share, offering developers a fixed 70% of billing revenue.

The last option is to create your own website and implement a payment mechanism through that, such as PayPal mobile, Dutch initiative èM! Payment[3], dial-in to premium landlines[4] 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.

3 empayment.com
4 daopay.com

In-App Payment

In-app payment 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 could implement your own payment mechanism. If you want to look at anything more than a one-off "full license" payment you have to think carefully about how, when and what your users will be willing to pay for and design your app accordingly.

This type of payment 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. Note, however, that some app stores do not allow third party payment options to be implemented inside your app. This is done to prevent you from using the app store for free distribution while avoiding payment of the store's revenue share.

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.

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, or with in-app payment offered directly by operators.

In Germany Deutsche Telekom, Vodafone and Telefónica/O2 became the first operators to launch in-app payment APIs that work cross-operator and enable billing directly to the phone bill of the user. 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. 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* 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 50 large mobile ad networks. Each network offers slightly different approaches and finding the one that monetizes your app's audience best may not be straightforward. There is no golden rule; you may have to experiment with a few to find the one that works best. However, for a quick start you might consider using a mobile ad aggregator, such as smaato[5], Madgic[6] or inneractive[7] as they tend to bring you better earnings by combining and optimizing ads from 30+ mobile ad networks. Most ad networks take a 30% to 50% share of advertising revenue and aggregators another 15% to 20% on top of that.

If your app is doing really well and has a large volume in a specific country you might consider selling ads directly to

5 smaato.net
6 www.madgic.com madgic.com
7 www.inner-active.com inner-active.com

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 mechanisms 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.

It may require some experimentation to find the right level and positions in which to place adverts.

Revenue Sharing

Revenue sharing with mobile operator for services built into your app is an emerging opportunity for developers, and one that is worth following. This monetization method lets developers build services such as SMS, MMS, location, advertising, customer profile and operator billing into their apps. With well-documented APIs that are free to use, revenue generated is split transparently between operator and app owner.

While BlueVia is currently the only developer community dedicated to this model, if its early adoption continues to grow, it may become a recognized business model for mobile operators.

Indirect Sales

Another option is to use your application to drive sales elsewhere.

Here you usually offer your app or website for free and then use mechanisms such as:

1. Affiliate programs: Promote third party or your own paid apps within a free app. See also MobPartner[8]. This can be considered a variation on mobile advertising
2. Data reporting: Track behavior and sell data to interested parties. Note that for privacy reasons you should not reveal any personal information, ensure all data is provided in anonymous, consolidated reports
3. Virtual vs. real world: Use your app as a marketing tool to sell goods in the real world. Typical examples are car apps, magazine apps and large brands such as McDonald's and Starbucks. Also coupon applications often use this business model

There is nothing to stop you from combining this option with any of the other revenue generation options if you wish, but take care that you do not give the impression of overly-intrusive promotions.

Component Marketplace

A Component Marketplace (CMP) provides another opportunity for developers to monetize their products to other developers and earn money by selling software components or white-labelled apps. A software component is a building block piece

8 mobpartner.com

of software, which provides a defined functionality, that is to be used by higher level software.

The typical question that comes up at this point is on how CMPs contrast to open source. As a user, open source is often free-of-charge. Source code must be provided and users have the right to modify the source code and distribute the derived work.

Some component providers require a license fee. They may provide full source code which enables the developer to debug into lower level code. Some CMPs support all models: Paid components with or without source code as well as free components with or without source code.

If you are a developer searching for a component, CMPs offer two major advantages: First, you don't have to open source your code just because you use software components. All open source comes with a license. Some licenses like the Apache are commercially friendly; others, such as AGPL and OSL, require you to open source your code that integrates with theirs. You might not want this. Secondly, CMPs provide an easy way to find and download components. You can spend days browsing open source repositories to find the right thing to use.

Component marketplaces have existed for decades now. The most prominent marketplace is for components for Visual Basic and .NET in the Windows community. Marketplaces such as componentOne and suppliers like Infragistics are well known in their domain. The idea of component marketplaces within the mobile arena is quite new. Recently, Developer Garden and Verious started into this domain[9] . Also, the idea to open the marketplace to semi-professional suppliers is new. In Developer Garden, you can register yourself and offer your component to the public – for free or as a paid offer.

9 www.developergarden.com/component-marketplace/

Now imagine you built a piece of code which could be seen as one or even several components. Next steps would include: providing extensive and detailed documentation. You might then also add several "hello-world" samples using your component. Combine the component in source, or precompiled with the documentation and the samples, and you are ready to offer it on the component marketplace. Since your customers will probably ask questions (the better the documentation is, the fewer questions you will get), it often also makes sense to offer an FAQ.

Choosing your Monetization Model

So with all these options what should your strategy be? It depends on your goals, let us look at a few:

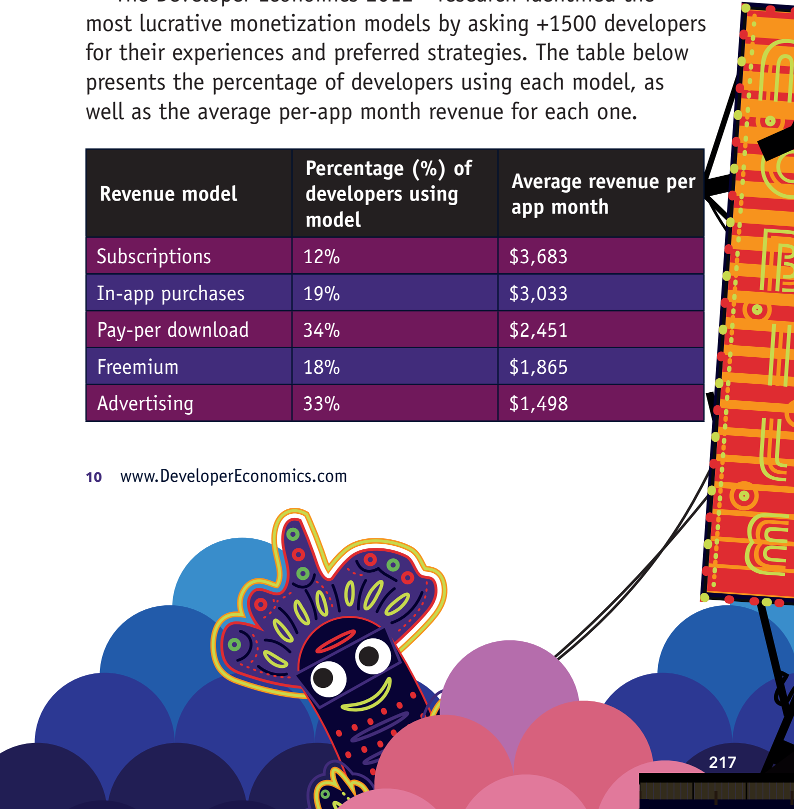
- Do you want a large user base? Consider distributing your application for free at first, then start adding mobile advertising or split between a free and paid version, when you have more than 100 thousand users worldwide
- Are you convinced users will be willing to buy your app immediately? Then sell it as PPD for $0.99, but beware while you might cash several thousand dollars per day it could easily be no more than a few hundred dollars per week if your assessment of your app is misplaced or the competition fierce
- 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

The Developer Economics 2012[10] research identified the most lucrative monetization models by asking +1500 developers for their experiences and preferred strategies. The table below presents the percentage of developers using each model, as well as the average per-app month revenue for each one.

Revenue model	Percentage (%) of developers using model	Average revenue per app month
Subscriptions	12%	$3,683
In-app purchases	19%	$3,033
Pay-per download	34%	$2,451
Freemium	18%	$1,865
Advertising	33%	$1,498

10 www.DeveloperEconomics.com



Appstores

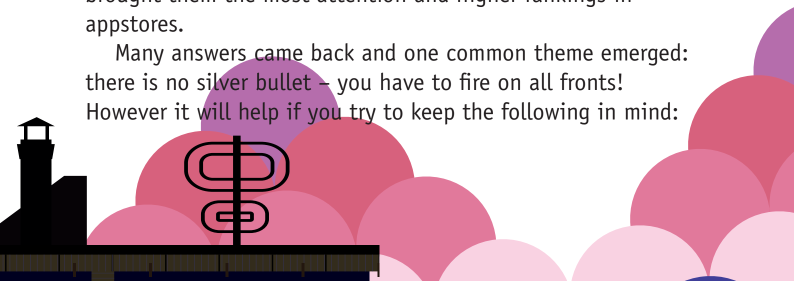
The flip side of revenue generation is marketing and promotion. The need might be obvious if you sell your application through your own website, but it is equally important when using a vendor's app store. Appstores are the curse and the blessing of mobile developers. On the bright side they give developers extended reach and potential sales exposure that would otherwise be very difficult to achieve. On the dark side the more popular ones now contain hundreds of thousands of apps, decreasing the potential to stand out from the crowd and be successful, leading many to compare the chances of appstore success to the odds of winning the lottery. So, here are a few tips and tricks to help your raise your odds.

Basic Strategies To Get High

The most important thing to understand about appstores is that they are distribution channels and not marketing machines. This means that while appstores are a great way to get your app onto users' devices, they are not going to market your app for you (unless you purchase premium positioning either through banners or list placings). You cannot rely on the app stores to pump up your downloads, unless you happen to get into a top-ten list. But do not play the lottery with your apps, have a strategy and plan to market your app.

We have asked many developers about the tactics that brought them the most attention and higher rankings in appstores.

Many answers came back and one common theme emerged: there is no silver bullet – you have to fire on all fronts! However it will help if you try to keep the following in mind:



- You need a kick ass app: it should be entertaining, easy to use and not buggy. Make sure you put it in the hands of users before you put it in a store.
- Polish your icons and images in the appstore, work on your app description, and carefully choose your keywords and category. If unsure of or unsatisfied with the results, experiment.
- Getting reviewed by bloggers and magazines is one of the best ways to get attention. In return some will be asking for money, some for exclusivity, and some for early access.
- Get (positive) reviews as quickly as possible. Call your friends and ask your users regularly for a review.
- If you are going to do any advertising, use a burst of advertising over a couple of days. This is much more effective than spending the same amount of money over 2 weeks, as it will help create a big spike, rather than a slow, gradual push.
- Do not rely on the traffic generated by people browsing the appstore, make sure you drive traffic to your app through your website, SEO and social media.

Multi-Store vs Single Store

With 120+ appstores available to developers, there are clearly many application distribution options. But the 20 minutes needed on average to submit an app to an appstore means you could be spending a lot of time posting apps in obscure stores that achieve few downloads. This is why a majority of developers stick to only 1 or 2 stores, missing out on a potentially huge opportunity but getting a lot more time for the important things, like coding! So should you go multi-store or not?

Multi-store	Single store
The main platform appstores can have serious limitations, such as payment mechanisms, penetration in certain countries, content guidelines.	90%+ of smartphone users only use a single appstore, which tends to be the platform appstore shipping with the phone
Smaller stores give you more visibility options (featured app)	Your own website can bring you more traffic than appstores (especially if you have a well-known brand)
Smaller stores are more social media friendly than large ones.	Many smaller appstores scrape data from large stores, so your app may already be there.
Operators' stores have notoriously strict content guidelines and can be difficult to get in, particularly for some types of apps.	For non-niche content, operator or platform stores may offer enough exposure to not justify the extra effort of a multi-store strategy.
Smaller stores may offer a wider range of payment or business model options, or be available in many countries.	Some operators' stores have easier billing processes, such as direct billing to a user's mobile account – leading to higher conversion rates.
Some developers report that 50% of their Android revenues come from outside of Android Market	iPhone developers only need 1 appstore

What Can You Earn?

One of the most common developer questions is about how much money they can make with a mobile app. It is clear that some apps have made their developer's millionaires, while others will not be giving up their day job anytime soon. According to a 2012 research by App-Promo.com[11], 59% of all app developers are not generating enough revenue to break even with development costs and 80% confirmed that the revenue generated with their most successful app was not enough to support a standalone business.

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.

When it comes to platforms, it is actually BlackBerry and iOS that top the per-app month revenue chart according to the Developer Economics research by VisionMobile. Android is third behind them, with Windows Phone bringing up the rear. The table below presents the average per app-month revenue for each platform according to the Developer Economics 2012 research.

11 app-promo.com/press-release-app-developers-get-a-wake-up-call-from-results-of-app-promos-first-annual-developer-survey/

Platform	Revenue per app-month
Android	$2,735
BlackBerry	$3,853
iOS	$3,693
Windows Phone	$1,234

Learn More

If you want to dig deeper into the topic of app marketing, check out the "Mobile Developer's Guide To The Parallel Universe" published by WIP. The 3rd edition has just been published and is available on their events and their website[12].

12 wipconnector.com

Epilogue

Thanks for reading this 12th 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. 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 or sponsor upcoming editions, please send your feedback to **developers@enough.de.**

If you are using Twitter, you are invited to follow us on **twitter.com/enoughsoftware** and spread the word about the project using the hashtag **#mdgg**

You can also order your hard copy of this publication if you write to *developers@enough.de* – we will only charge for the postage costs.

About the Authors

Mostafa Akbari / bitstars

Mo worked in software engineering and human interaction research the past few years. He is involved in green mobility projects. Now Mo has started a spinoff with Simon Heinen out of the RWTH Aachen University for augmented reality research and development. He focuses on AR interactions with personal location-based data and on computer vision. His passion for mix reality games is based on his passion for board games and geocaching.

@mosworld **www.bitstars.com**

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 in-house as UI and UX designer for consumer facing products on mobile and desktop. Apart from thinking through and drawing screens she also does illustration and enjoys living in London. **www.alfutka.net**

Andrej Balaz / Enough Software

As a graduate of the University of the Arts Bremen, Andrej focuses on UI, UX and visual design for mobile applications. He is also in charge of the layout and design of this guide. Currently he is involved in the research and concept creation for the opportunity spaces of art and music festivals in the pursuit of his Master's Degree in Digital Media Design.

@abiozzUI **www.enough.de** **behance.net/andrejbalaz**

Davoc Bradley / Redware

Davoc has been working as a software engineer since 1999 specializing in architecture and design of high usage web and mobile systems. Most recently he was behind the architecture, design and development of Redware's gold award winning Mobile Application Management offering "Redsource". Davoc is also a keen musician, avid cricket fan and loves travelling.

@davocbradley **www.redware.co.uk**

Richard Bloor / Sherpa Consulting Ltd

Richard has been writing about mobile applications development since 2000. He has contributed to popular websites, such as AllAboutSymbian.com, but now focuses on assisting companies in creating resources for developers. Richard brings a strong technical background to his work, having managed development and testing on a number of major IT projects, including the Land Information NZ integrated land ownership and survey system. When not writing about mobile development, Richard can be found regenerating the native bush on his property north of Wellington, New Zealand.

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 outside the USA 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

Bob Heubel / Immersion Corp.

Bob is a haptic technology evangelist with Immersion Corporation who specializes in assisting developers implement what is known as force-feedback, tactile-feedback or rumble-feedback effects. He has spent more than ten years working with developers, carriers, OEMs and ODMs to design and implement these sensations aimed at improving both gaming and user interaction experiences. Bob graduated from UC Berkeley in 1989 with a BA in English Literature.

@bobheubel **www.immersion.com**

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.enough.de** **www.ovidiuiliescu.com**

Gary Johnson / Sharkfist Software, LLC

Gary is currently contracting for mobile development across the board – Windows Phone, iOS and Android. His past experience includes deep expertise in .NET, Silverlight and WPF. He has a strong passion for all things mobile as well as creating great UI and UX.

Alex Jonsson / MoSync

Alex likes anything mobile, both apps and web technologies as well as cleverly connecting physical stuff to digital stuff. He holds a Doctorate in Computer Science and still gives lectures now and then. Alex has an eclectic urge to create new values by finding new combinations of things, transferring knowledge between disciplines and exploiting aspects of communication and media with the aim of bringing people together. Alex is CTO at MoSync Inc.

@dr_alexj **www.mosync.com**

Matos Kapetanakis / VisionMobile

As marketing manager of VisionMobile his activities include managing the VisionMobile website and blog, as well as coming up with the concepts and marcoms for the illustrations and infographics published by the company. Matos is also the project manager of the Developer Economics research series, as well as other developer research projects.

www.visionmobile.com

Michael Koch / Enough Software

Michael has developed software since 1988 and joined the development team at Enough Software in 2005. He holds the position of CTO. He has led numerous mobile app development projects (mainly for Java ME, 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.

@linux_pinguin **www.enough.de**

Daniel Kranz / Sevenval

Daniel is a multi-channel strategist with consultancy, agency and tech background. Previously a project manager at one of the leading advertising agencies, he now works as a mobile solution consultant advising brands on how to integrate mobile as part of their overall digital strategy.

www.sevenval.com **www.fitml.com**

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.

@seraandroid & @PayPalEuroDev
www.timmesserschmidt.de

Gary Readfern-Gray / RNIB

Gary is an Accessibility specialist working for the Royal National Institute of the Blind. Located in the Innovation Unit, he has a passion for the mobile space and particularly for enabling accessible app development across a range of platforms by engaging with developer communities.
www.rnib.org.uk

Alexander Repty

Alexander has been developing software for Mac OS X since 2004. When the iPhone SDK was released in 2008, he was among the first registered developers for the program. As an employee of Enough Software, he worked on a number of apps, one of which was featured in an Apple TV commercial. He has written a series of articles on iPhone development. As of April 2011, he started his own business as an independent software developer and contractor.
@arepty **www.alexrepty.com**

Marcus Ross

Marcus is a freelance developer and trainer. After 10 years of being an employee in several companies, he is now doing SQL-BI Projects and everything mobile and cross-platform. He is a regular author in the German magazine "mobileWebDeveloper". In his spare time, he is often seen at conferences, speaking on mobile subjects and JavaScript. He also writes articles, books & tweets on mobile development.
@zahlenhelfer **www.zahlenhelfer-consulting.de**

Thibaut Rouffineau / WIP

Community and passion builder with a mobile edge, for the past 5 years Thibaut has been working to move the mobile developer community towards greater openness and exchange. Thibaut is VP for Developer Partnerships at WIP, the organizer of Droidcon London. He is an enthusiastic speaker on mobile topics and has been heard around the world.

@thibautr **www.wipconnector.com**

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 *m.wauwee.com*. Serving almost 1,000,000 members, Michel was supported by a team of Symbian, iPhone, BlackBerry and Android specialists at headquarters in Amsterdam. *m.wauwee.com* was acquired by MobiLuck.

www.appvalley.nl

Shailen Sobhee / Enough Software

Shailen is a recent graduate in Electrical Engineering and Computer Science at Jacobs University Bremen. He is currently pursuing his Master's degree in Computational Science and Engineering at the Technical University in Munich. Shailen joined Enough Software in July 2012 as an intern and has since been developing cutting-edge Android software. With a strong computer science background, Shailen finds interest in new technologies that can potentially have a huge positive impact on the future. Currently he is developing and researching on new solutions based on NFC technologies.

www.enough.de

Marco Tabor / Enough Software

Marco is responsible for PR, sales and much more at Enough Software. He coordinates this project, taking, as well, the responsibility of finding sponsors and merging the input provided by the mobile community.

@enoughmarco **www.enough.de**

Ian Thain / SAP

Ian is a Mobile Evangelist at SAP, though he started 12 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.

@ithain **ianthain.ulitzer.com** **www.sap.com**

Robert Virkus / Enough Software

Robert has been working in the mobile space since 1998. He experienced Java fragmentation first hand when developing and porting a mobile client on the Siemens SL42i, the first mass market phone with an embedded Java VM. After this experience he launched the Open Source J2ME Polish project in 2004. J2ME Polish helps developers overcome device fragmentation. He is the founder and CEO of Enough Software, the company behind J2ME Polish, many mobile apps, and this book.

@robert_virkus **www.enough.de** **www.j2mepolish.org**

an initiative by:

ENOUGH SOFTWARE
www.enough.de

wip wireless industry partnership
www.wipconnector.com

printing sponsors:

SAP®

BlackBerry®

PayPal™ | Developer

A NON-COMMERCIAL, COMMUNITY-DRIVEN OVERVIEW ON MOBILE TECHNOLOGIES FOR DEVELOPERS AND DECISION-MAKERS.

Daniel Hudson, www.webtechman.com
A spectacular piece of work! You will be astonished by how incredibly fast you can establish your presence in the mobile market with the simple steps explained in this guide.

Monika Lischke, Community Manager, Intel AppUp developer program
Extremely helpful content, also for non-developers. And the design is nothing but fantastic!