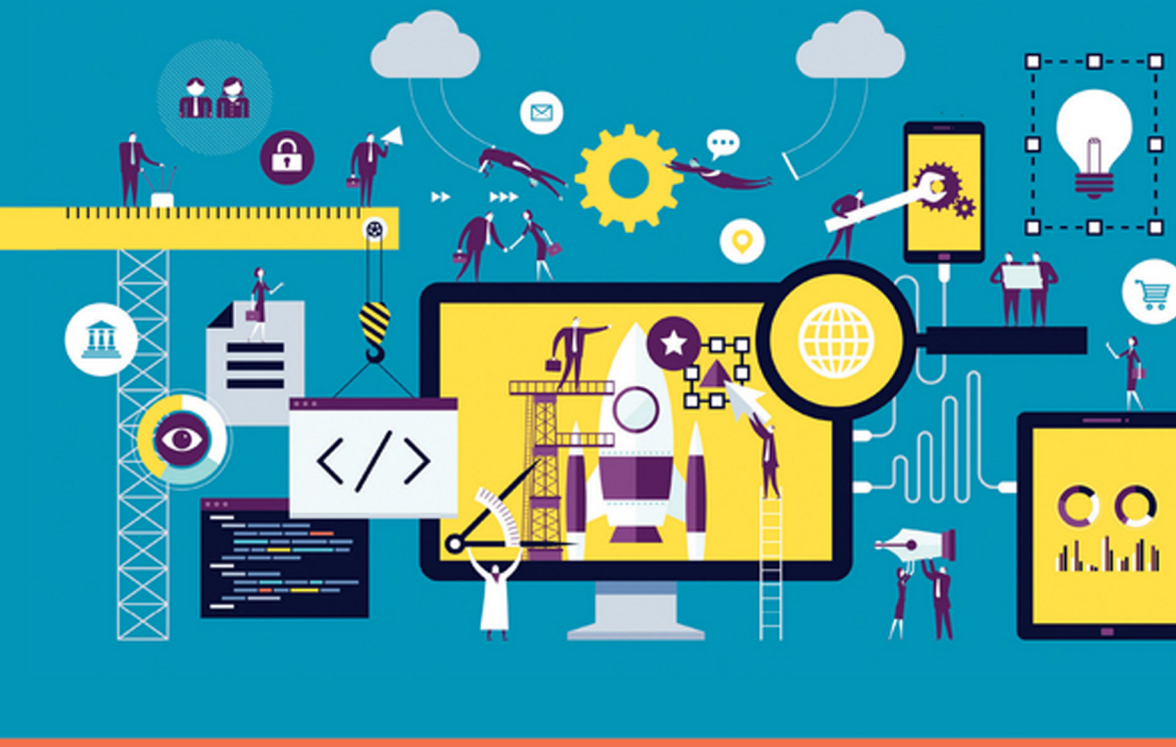

ROCKY HECKMAN

DESIGNING PLATFORM INDEPENDENT MOBILE APPS AND SERVICES

IEEE PRESS

IEEE computer society

WILEY

DESIGNING PLATFORM INDEPENDENT MOBILE APPS AND SERVICES

About IEEE Computer Society

IEEE Computer Society is the world's leading computing membership organization and the trusted information and career-development source for a global workforce of technology leaders including: professors, researchers, software engineers, IT professionals, employers, and students. The unmatched source for technology information, inspiration, and collaboration, the IEEE Computer Society is the source that computing professionals trust to provide high-quality, state-of-the-art information on an on-demand basis. The Computer Society provides a wide range of forums for top minds to come together, including technical conferences, publications, and a comprehensive digital library, unique training webinars, professional training, and the TechLeader Training Partner Program to help organizations increase their staff's technical knowledge and expertise, as well as the personalized information tool myComputer. To find out more about the community for technology leaders, visit http://www.computer.org.

IEEE/Wiley Partnership

The IEEE Computer Society and Wiley partnership allows the CS Press authored book program to produce a number of exciting new titles in areas of computer science, computing, and networking with a special focus on software engineering. IEEE Computer Society members continue to receive a 15% discount on these titles when purchased through Wiley or at wiley.com/ieeecs.

To submit questions about the program or send proposals, please contact Mary Hatcher, Editor, Wiley-IEEE Press: Email: mhatcher@wiley.com, Telephone: 201-748-6903, John Wiley & Sons, Inc., 111 River Street, MS 8-01, Hoboken, NJ 07030-5774.

DESIGNING PLATFORM INDEPENDENT MOBILE APPS AND SERVICES

ROCKY HECKMAN

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Names: Heckman, Rocky, author.
Title: Designing platform independent mobile apps and services / Rocky Heckman.
Description: Hoboken, New Jersey : John Wiley & Sons, Inc., [2016] | Includes index.
Identifiers: LCCN 2016009419| ISBN 9781119060147 (cloth) | ISBN 9781119060185 (epub) | ISBN 9781119060154 (Adobe PDF)
Subjects: LCSH: Mobile computing. | Cell phones–Programming. | Mobile apps.
Classification: LCC QA76.59 .H43 2016 | DDC 005.25–dc23 LC record available at https://lccn.loc.gov/2016009419

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Thank you to all my friends who finally convinced me to write a book. Most of all, thank you to my wife Stefanie, and my two beautiful girls Elyssia and Keiralli for not only pushing me to finish, but putting up with all the time I spent doing it. I love you all very much.
Look Dad, I did it!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

I WAS WONDERING how to start this preface, and it occurred to me that writing one is backwards. After all, this is the text before the book, but you write it after the book is finished. I suppose that is analogous to how we have written software for a long time. We figure out what it is we are trying to do usually though creating systems that do what we think we want them to do; then we go back and write the documentation about what the system actually does.

This book, in large part, is aimed at helping to make that a more harmonious effort. Technology is moving so fast now that often we find ourselves trying to create mobile apps and services in a very reactionary manner. We tend to be on the back foot and playing catch up most of the time. If we could just take the time to sharpen the proverbial axe, we'd be able to get more accomplished, faster, with a lot less hair pulling.

I suppose over the past couple decades of doing this, I've seen that pattern time and time again. But there are also some good habits and patterns that I've seen along the way that in some respect were ahead of their time. Service-oriented architecture, for example, was a great idea for connecting the myriad of systems we've had inside organizations with simple, easy-to-use interfaces. In fact, this tried-and-true pattern, or collection of patterns if you will, is more relevant today in our commodity cloud computing, mobile app world than ever before.

This book is designed to provide some high level architectural guidance on how to design modern mobile apps and services so that they can adapt as technology inevitably changes. Of course, we start off with a brief history of our mobile computing explosion, and take a look at attempts to create cross platform apps and technology stacks.

Then I want to introduce what hopefully has become an obvious application stack. While we have been fairly fixated on a *N*-tier stack, where *N* usually equaled three, to truly futureproof our architectures, we really need two more clearly defined layers to provide us an abstraction boundary which insulates our code from changes in external client technology, as well as the rapidly changing data storage technologies we use today.

Once we have our layers sorted out, we'll have a look at various patterns of application development and how they apply to this layering system to create performant and resilient services for making powerful mobile applications.

I hope that you find this guidance useful. Perhaps it will make you think of things in ways you hadn't before, or validate thinking you've already implemented. In any case, I hope it prevents you from having to operate in a reactionary manner to the rapid changes of our modern computing world and lets you get on the front foot

so you can focus on creating great apps and services instead of retooling everything because a new phone hit the market.

Target Audience

This book is for anyone who is responsible for the design, architecture, and development of modern mobile apps and the services that support them. I've written this book with futureproofing in mind. Ideally, the architectures and patterns in this book will provide you with an approach that will futureproof your designs.

By following this guidance, you should be able to create mobile app services that you can adapt, modify, update, change, or integrate without disrupting your mobile apps, or your teams. You should be able to deploy new services, change existing services, and add new client apps all without disturbing any of the running systems.

Most of all, you should be able to adapt services and apps based on this guidance to any new mobile platform that comes along. This will greatly increase your code reuse, make your teams much more efficient, and make your organization adaptable to the ever-changing mobile app landscape.

ACKNOWLEDGMENTS

These kinds of things don't happen without a lot of people in the background pushing, pulling, helping, and sometimes simply nodding and smiling. I would like to thank Chris Bright for encouraging me and allowing me the time to put this together. I'd also like to thank Andrew Coates and Dave Glover for letting me harass them with ideas, and "what if" questions all the time.

Most important of all, I need to thank my wife Stefanie Heckman, and my two girls Elyssia and Keiralli. They not only encouraged me to finish, but were patient with me, and gave me the time to keep typing away. I think, in the end, their love and enthusiasm are what really got this book over the line. So if you like it, don't forget to thank them too.

CHAPTER 1

THE MOBILE LANDSCAPE

1.1 INTRODUCTION

When the idea of reaching people first struck home in the dark ages, we wanted to find ways for people to use and pay for our services. We had to find a way to let people know these services existed. In early days there were town criers, then during the industrial revolution when we could reproduce and distribute text to a largely literate audience, we had broadsheets. Then came the catalogue where mercantile companies would list their wares for sale. Once we heard the first radio waves, one of the first things they did was to sell advertising on the radio. This graduated to television advertising. Then along came the Internet. Everyone had to get themselves a website and would put their website address in their print, radio, and TV ads. Along came Facebook and everyone created a Facebook page for their companies.

Now, everyone wants to have an app for their customers to download. These apps go with customers wherever they are and provide instant interaction between consumer and supplier. We can push advertising into them, take orders through them, keep in touch with friends and relatives, and of course play games, listen to music and watch videos all in the palms of our hands. These experiences require devices, operating systems, and apps, all of which require software companies, architects, and developers to produce them. Unfortunately, these devices and operating systems often change.

In today's computing world, there is one thing you can be sure of; the leading operating system (OS) platform will change. As recently as 7 years ago, Microsoft Windows Mobile was the leading smartphone platform and tablet computers, while mobile, were large and clunky and ran full versions of the Windows XP and Windows 7 OS. Then came Blackberry which took a lot of market share from Windows Mobile. But that only lasted until the iPhone came along in 2007 and we went from a feature phone dominated world to a smartphone dominated one. This set a new benchmark and became the leading mobile computing platform. In the same year, the Open Handset Alliance re-released Linux-based Android-powered smartphones. Then in 2010, Google launched its Android-based Nexus series of devices. By 2011, Android-powered smartphones made up the majority of mobile OS-powered smartphones shooting past the iPhone.

While phones were taking off, in 2010 Apple released the iPad. Tablet computing was not new and in fact Microsoft and its Original Equipment Manufacturers

Designing Platform Independent Mobile Apps and Services, First Edition. Rocky Heckman.

OEM partners had been trying to sell tablet computers since 2003. However, the iPad's sleek design brought tablet computing to the masses despite the clumsy and restrictive iOS operating system. This opened up the tablet computing market which Android was well suited for. After the iPad's initial success, by 2013 Google's Android-powered tablets had overtaken iPads as the tablet of choice. Additionally, although lagging considerably behind, Microsoft has re-created itself to make a run in the mobile and tablet computer markets as well. With Microsoft's massive install base, and very large developer ecosystem, they are likely to challenge Apple and Google in the mobile and tablet space eventually. With Windows 10 released as a free upgrade for over 1.5 billion eligible devices [1], it is likely to be the most common cross platform OS. That is, over half of Gartner's predicted 2,419,864,000 devices shipped into the market in 2014 [2]. Overnight the app ecosystem market leader could change again.

What this means for software developers, independent software vendors (ISVs), hobbyist app developers, and online service providers is that every few years they will have to retarget their efforts for a new platform, new development languages, new development tools, new skills, and new ways of thinking. This is not an attractive proposition for anyone. However, due to the success of the iPhone and iPad, software developers were willing to re-skill and even purchase proprietary hardware just to be able to develop applications for the new platforms. Then when Android devices surpassed the Apple devices, these same developers painfully went through the whole process again. Developers were forced to maintain three or more separate and complete codebases. This is the problem that Platform-Independent Delivery of Mobile Apps and Services solves.

If you are not planning a platform-independent strategy, you will likely be an ex-company in 3–5 years. Due to the rapid change of the consumer and enterprise mobile computing landscape, software developers must be able to adapt to new platforms, devices, and services before their competition. While cross-platform goes a long way toward this goal, it is still cumbersome and tends to lag behind a more platform-independent strategy. While it is not practical to get completely away from device-specific app code, the more you can move off of the device and put into a reusable back-end service, the less code you have to write and maintain when a new OS version or a new platform comes along. In this book we will examine strategies to do this, and provide future proof foundations to support changes in the computing landscape down the road.

Disclaimer: This book was written in late 2014. Everything in it was accurate at that time. If you are reading this in 2025, expect that a few things have changed. Just keep this in mind as we go through this so I don't have to keep writing "At the time of this writing…."

1.2 PREVIOUS ATTEMPTS AT CROSS-PLATFORM

1.2.1 Java

"Write once, run everywhere" was a slogan developed by Sun Microsystems which promised cross-platform capability for Java applications supported by Duke, Java's Mascot shown in Figure 1.1. This gained significant traction in the mid to late 1990s.

Figure 1.1 Duke

In the beginning of this era, the promise seemed legitimate. You could write the Java code once, package up your Java byte code in .jar files and run them on any system that had the Java Virtual Machine (JVM) interpreter. It worked so well that there are even C to Java compilers so your C applications can run with the same cross-platform reach that Java had.

The promise was that you could write code like this:

```
class CrossPlatformApp {
    public static void main(String[] args) {
        System.out.println("I am omnipresent!");
          // Display the string.
    }
}
```

And it would run on every computer and device that ran Java without compiling multiple versions for each target device. All you had to do was make sure that the target device had the correct version of the JVM installed on it.

This worked fine until various vendors started creating their own versions of the JVM to run on their platforms. By 2014, more than 70 different JVMs [3] had been created that could run Java applications, for the most part. The catch was that they were each slightly different.

If we take the Sun JVM to be the standard, some of these other JVMs were better, and most were worse, at interpreting Java byte code. Some of them such as the IBM J9 (http://en.wikipedia.org/wiki/IBM_J9), the Azul Zing JVM (http://en.wikipedia.org/wiki/Azul_Systems), and the Microsoft JVM (http://en.wikipedia.org/wiki/Microsoft_Java_Virtual_Machine) were better

and faster than the original Sun JVM. They even went so far as to add extra features and some constructs that were more familiar to traditional C/C++ programmers in order to make the transition easier for them.

While this seemed fantastic at the time, because it meant every platform vendor had a JVM to run Java, they weren't all the same. So what may work on the Sun JVM may not work on the Microsoft or IBM implementation. Even though some of these implementations such as the 1998–1999 Microsoft JVM outperformed the Sun version, they weren't entirely compatible with the Java 1.1 standard. This lead to Sun suing Microsoft and other JVM vendors in an attempt to try to defragment the Java playing field. The result was these other vendors stopped supporting their proprietary versions of the JVM and true high-performance, cross-platform capability for Java applications started to deteriorate.

This is a trade-off that you see repeatedly in cross-platform development. There has always been a compromise between running on many different devices, and getting as close to the hardware as possible for fast execution. It's the nature of computers. Each device may have slightly different hardware running the code. This means that the operating system and CPU may understand different instructions on each device. Java tried to solve this with the JVM. Different JVMs are written for the different environments, and they provide an abstraction layer between your Java code, and the nuances of the underlying hardware. The problems arise when one JVM interpreted the incoming Java code slightly differently than the next one and the Java dream becomes fragmented.

While Java is still widely used for applications, there are many versions of it depending on what kind of applications you are writing. There are four primary versions of Java that are supported by Sun.

- Java Card for smartcards.
- Java Platform, Micro Edition (Java ME) for low powered devices
- Java Platform, Standard Edition (Java SE) for standard desktop environments
- Java Platform, Enterprise Edition (Java EE) for large-scale distributed internal enterprise applications

All of them require a very standards adherent JVM to be installed on the target machine for them to run. Often the JVM can be packaged up with the application deployment, but the dependence on the JVM and specific versions of the JVM have made cross-platform Java apps troublesome. This is largely because you can never be sure of the JVM on the target device.

This is a common issue with most interpreted languages such as Java, Python, Ruby, .NET and any other language that is Just-In-Time compiled and run in a virtual environment or through a code interpreter. These kinds of things also reduce the speed of the applications because everything is interpreted on the fly and then translated for the CPU rather than being compiled down into Assembly or CPU level instructions which are executed by the CPU natively. This is why C and C++ and similar languages are referred to as native languages.

So while Java was a very good attempt at write once run anywhere, it fell short due to its dependency on the JVM. It still has a large install base and works very

well in many web app scenarios. It is also the primary app development language for Android-based devices which at the time of this writing was the world's leading mobile device operating system. Java can also be used to create apps for Apple's iOS-based devices through systems such as the Oracle ADF Mobile Solution [4, 5]. However, the vast majority of iOS targeted apps are written in Objective-C using Apple's Xcode environment. Due to the difficulty of developing with Objective-C, Apple introduced a new language called Swift for the iOS platform to help combat the hard translation form Java or C# for iOS developers and to improve the performance over Objective-C apps. At the time of this writing, Java did not work on the Windows Modern apps or Windows Phone platforms. Java does still work in the Windows Pro full x86 environment.

Java is perhaps the closest the industry has come to write once run anywhere. But it has been plagued by spotty JVM support, and a push toward more proprietary development for iOS and Windows to get the speed and integration to a more seamless state.

1.2.2 Early Web Apps

On August 6, 1991 the first website was created by Tim Berners-Lee at CERN (http://info.cern.ch/hypertext/WWW/TheProject.html). Ever since then, we've been pushing web browsers beyond their intended limits. From their humble beginnings as static pages of information to the preeminent source of all information and social interaction, websites and web apps have become ubiquitous in our connected world. It was inevitable that web access from every computer would lead to web apps being seen as the next great cross-platform play.

Web apps have been a popular attempt to run anywhere. All you need is a web browser on whatever device you have and you can use web apps. Well, that's the idea anyway. In reality this proved much more difficult than anyone hoped. Prior to HTML 5 you naturally had HTML 4. HTML 4 was still largely just a markup language designed to handle content formatting. Web pages displayed in the browsers were largely static text and images. Then early browsers such as Netscape and Internet Explorer 3 incorporated a JVM to interpret the Java code in the web pages. Web server software such as Apache and Internet Information Services could also run server-side Java code and send the product of the code back to the browser as an HTML page.

This worked pretty well, up until Sun sued Microsoft and they stopped including the Microsoft JVM with Internet Explorer. Since at the time it had become the world's most popular browser, that was a problem for Java-based web apps. It forced users to manually install a JVM from Sun which was an extra step most people weren't overly fond of.

This resulted in some interesting changes. Netscape produced its own web programming language called LiveScript in 1995, which it then changed the name to JavaScript when it introduced Java support in Netscape 2 in 1996. Meanwhile in the same year Microsoft produced Active Server Pages (ASP) and in an attempt to get around the Java JVM problem, it also included VBScript for the coding portion in ASP. JavaScript pretty much won the client-side scripting battle when Microsoft included support for it in Internet Explorer 3 but had to call its version JScript.

JavaScript became adopted as a standard known as ECMAScript which is in its fifth edition (5.1) released in June 2011.

In order to do interesting things with web apps, we needed to do things that HTML 4 simply couldn't do on its own. So one of the first things that was built into web browsers was a JavaScript interpreter. Now you could run scripts in web pages that could do things like display today's date, manipulate text in text boxes, and rotate pictures. This was nice, but in the days of Mosaic/Mozilla, Netscape, and Internet Explorer 3, it was really pushing the envelope.

To get a bit more out of the web apps, people started developing plugins for web browsers for things like audio and video. Macromedia introduced Flash in 1996 and it opened up all kinds of new opportunities to do very advanced graphics in a web browser through the Adobe Flash Player. By 2000, Flash was everywhere and even used to produce some animated TV commercials and 2D programs [6]. Around the same time in 2007, Microsoft introduced Silverlight which was a competing technology to Flash and offered audio, video, and graphics for web apps.

At the time, HTML had been reduced down to something like the following:

```
<HTML>
<HEAD></HEAD>
<BODY>
…
</BODY>
</HTML>
```

Everything between the <BODY>…</BODY> tags were references to JavaScript and plugins of some sort that offered extended capabilities that were not part of the HTML4 specification. This included embedded audio, video, and pluggable content.

The core of the problem was that while HTML was an open standard that everyone understood and agreed on, things like Silverlight and Flash were not. This lead to controversy about its use and widespread adoption due to the dependency on proprietary technologies. In fact David Meyer quoted Tristan Nitot of Mozilla as saying:

> "*You're producing content for your users and there's someone in the middle deciding whether users should see your content," [7]*

This sentiment essentially created a mistrust of proprietary technologies that started developers looking for standards bodies to create web standards that could fill the voids that things like Silverlight and Flash handled.

Although these technologies are still prevalent today, they met with some resistance in the mobile computing era. Some of it was due to Apple initially not allowing Flash to operate on its iOS platform. While this was fixed by allowing Adobe AIR apps to run on an iPhone which wrapped Flash content, it was enough for Adobe to re-evaluate its position on Flash and to withdraw support for Flash on mobile devices in 2011 unless it is embedded in Adobe AIR applications. Instead, they plan to "aggressively contribute to HTML5." [8]