1

Introducing Android

The mobile development community is at a tipping point. Mobile users demand more choice, more opportunities to customize their phones, and more functionality. Mobile operators want to provide value-added content to their subscribers in a manageable and lucrative way. Mobile developers want the freedom to develop the powerful mobile applications users demand with minimal roadblocks to success. Finally, handset manufacturers want a stable, secure, and affordable platform to power their devices. Upuntil now single mobile platform has adequately addressed the needs of all the parties.

Enter Android, which is a potential game-changer for the mobile development community. An innovative and open platform, Android is well positioned to address the growing needs of the mobile marketplace.

This chapter explains what Android is, how and why it was developed, and where the platform fits in to the established mobile marketplace.

A Brief History of Mobile Software Development

To understand what makes Android so compelling, we must examine how mobile development has evolved and how Android differs from competing platforms.

Way Back When

Remember way back when a phone was just a phone? When we talked on fixed landlines? When we ran for the phone instead of pulling it out of our pocket? When we lost our friends at a crowded ballgame and waited around for hours hoping to reunite? When we forgot the grocery list (Figure 1.1) and had to find a payphone or drive back home again?

Those days are long gone. Today, commonplace problems like these are easily solved with a one-button speed dial or a simple text message like "WRU?" or "20?" or "Milk and?"

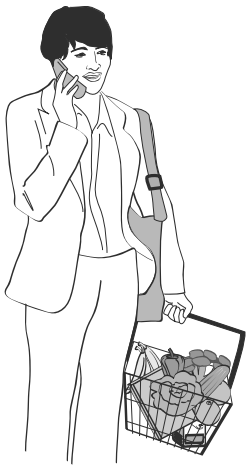

Figure 1.1 Mobile phones have become a crucial shopping accessory.

Our mobile phones keep us safe and connected. Nowadays, we roam around freely, relying on our phones not only to keep in touch with friends, family, and coworkers, but also to tell us where to go, what to do, and how to do it. Even the most domestic of events seem to revolve around my mobile phone.

Consider the following true, but slightly enhanced for effect, story:

Once upon a time, on a warm summer evening, I was happily minding my own business cooking dinner in my new house in rural New Hampshire when a bat swooped over my head, scaring me to death.

The first thing I did—while ducking—was pull out my cell and send a text message to my husband, who was across the country at the time: "There's a bat in the house!"

My husband did not immediately respond (a divorce-worthy incident, I thought at the time), so I called my Dad and asked him for suggestions on how to get rid of the bat.

He just laughed.

Annoyed, I snapped a picture of the bat with my phone and sent it to my husband and my blog, simultaneously guilt-tripping him and informing the world of my treacherous domestic wildlife encounter.

Finally, I Googled "get rid of a bat" and followed the helpful do-it-yourself instructions provided on the Web for people in my situation. I also learned that late August is when baby bats often leave the roost for the first time and learn to fly. Newly aware that I had a baby bat on my hands, I calmly got a broom and managed to herd the bat out of the house.

Problem solved—and I did it all with the help of my trusty cell phone, the old LG VX9800.

My point here? Mobile phones can solve just about *anything*—and we rely on them for *everything* these days.

You notice that I used half a dozen different mobile applications over the course of this story. Each application was developed by a different company and had a different user interface. Some were well designed; others not so much. I paid for some of the applications, and others came on my phone.

As a user, I found the experience functional, but not terribly inspiring. As a mobile developer, I wished for an opportunity to create a more seamless and powerful application that could handle all I'd done and more. I wanted to build a better bat trap, if you will.

Before Android, mobile developers faced many roadblocks when it came to writing applications. Building the better application, the unique application, the competing application, the hybrid application, and incorporating many common tasks such as messaging and calling in a familiar way were often unrealistic goals.

To understand why, let's take a brief look at the history of mobile software development.

"The Brick"

The Motorola DynaTAC 8000X was the first commercially available cell phone. First marketed in 1983, it was 13 x 1.75 x 3.5 inches in dimension, weighed about 2.5 pounds, and allowed you to talk for a little more than half an hour. It retailed for $3,995, plus hefty monthly service fees and per-minute charges.

We called it "The Brick," and the nickname stuck for many of those early mobile phones we alternatively loved and hated. About the size of a brick, with a battery power just long enough for half a conversation, these early mobile handsets were mostly seen in the hands of traveling business execs, security personnel, and the wealthy. First-generation mobile phones were just too expensive. The service charges alone would bankrupt the average person, especially when roaming.

Early mobile phones were not particularly full featured. (Although, even the Motorola DynaTAC, shown in Figure 1.2, had many of the buttons we've come to know well, such as the SEND, END, and CLR buttons.) These early phones did little more than make and receive calls and, if you were lucky, there was a simple contacts application that wasn't impossible to use.

These first-generation mobile phones were designed and developed by the handset manufacturers. Competition was fierce and trade secrets were closely guarded. Manufacturers didn't want to expose the internal workings of their handsets, so they usually developed the phone software in-house. As a developer, if you weren't part of this inner circle, you had no opportunity to write applications for the phones.

It was during this period that we saw the first "time-waster" games begin to appear. Nokia was famous for putting the 1970s video game *Snake* on some of its earliest monochrome phones. Other manufacturers followed, adding games like Pong, Tetris, and Tic-Tac-Toe.

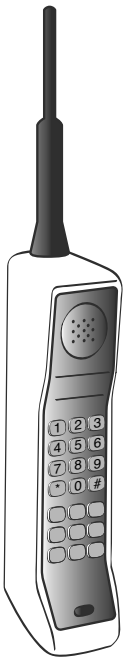

Figure 1.2 The first commercially available mobile phone: the Motorala DynaTAC.

These early phones were flawed, but they did something important—they changed the way people thought about communication. As mobile phone prices dropped, batteries improved, and reception areas grew, more and more people began carrying these handy devices. Soon mobile phones were more than just a novelty.

Customers began pushing for more features and more games. But, there was a problem. The handset manufacturers didn't have the motivation or the resources to build every application users wanted. They needed some way to provide a portal for entertainment and information services without allowing direct access to the handset.

And what better way to provide these services than the Internet?

Wireless Application Protocol (WAP)

It turned out allowing direct phone access to the Internet didn't scale well for mobile.

By this time, professional Web sites were full color and chock full of text, images, and other sorts of media. These sites relied on JavaScript, Flash, and other technologies to enhance the user experience and were often designed with a target resolution of 800x600 pixels and higher.

When the first clamshell phone, the Motorola StarTAC, was released in 1996, it merely had a LCD 10-digit segmented display. (Later models would add a dot-matrix type

display.) Meanwhile, Nokia released one of the first slider phones, the 8110—fondly referred to as "The Matrix Phone," as the phone was heavily used in films. The 8110 could display four lines of text with 13 characters per line. Figure 1.3 shows some of the common phone form factors.

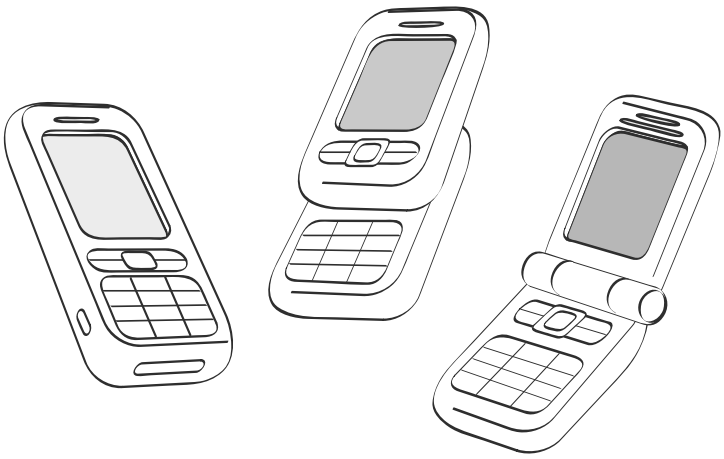

Figure 1.3 Various mobile phone form factors: the candy bar, the slider, and the clamshell.

With their postage stamp-sized low-resolution screens and limited storage and processing power, these phones couldn't handle the data-intensive operations required by traditional Web browsers. The bandwidth requirements for data transmission were also costly to the user.

The Wireless Application Protocol (WAP) standard emerged to address these concerns. Simply put, WAP was a stripped-down version of HTTP, which is the backbone protocol of the Internet. Unlike traditional Web browsers, WAP browsers were designed to run within the memory and bandwidth constraints of the phone. Third-party WAP sites served up pages written in a markup language called Wireless Markup Language (WML). These pages were then displayed on the phone's WAP browser. Users navigated as they would on the Web, but the pages were much simpler in design.

The WAP solution was great for handset manufacturers. The pressure was off—they could write one WAP browser to ship with the handset and rely on developers to come up with the content users wanted.

The WAP solution was great for mobile operators. They could provide a custom WAP portal, directing their subscribers to the content they wanted to provide, and rake in the data charges associated with browsing, which were often high.

Developers and content providers didn't deliver. For the first time, developers had a chance to develop content for phone users, and some did so, with limited success.

Most of the early WAP sites were extensions of popular branded Web sites, such as CNN.com and ESPN.com, looking for new ways to extend their readership. Suddenly phone users accessed the news, stock market quotes, and sports scores on their phones.

Commercializing WAP applications was difficult, and there was no built-in billing mechanism. Some of the most popular commercial WAP applications that emerged during this time were simple wallpaper and ringtone catalogues, allowing users to personalize their phones for the first time. For example, the users browsed a WAP site and requested a specific item. They filled out a simple order form with their phone number and their handset model. It was up to the content provider to deliver an image or audio file compatible with the given phone. Payment and verification were handled through various premium-priced delivery mechanisms such as Short Message Service (SMS), Enhanced Messaging Service (EMS), Multimedia Messaging Service (MMS), and WAP Push.

WAP browsers, especially in the early days, were slow and frustrating. Typing long URLs with the numeric keypad was onerous. WAP pages were often difficult to navigate. Most WAP sites were written once for all phones and did not account for individual phone specifications. It didn't matter if the end-user's phone had a big color screen or a postage stamp-sized monochrome one; the developer couldn't tailor the user's experience. The result was a mediocre and not very compelling experience for everyone involved.

Content providers often didn't bother with a WAP site and instead just advertised SMS short codes on TV and in magazines. In this case, the user sent a premium SMS message with a request for a specific wallpaper or ringtone, and the content provider sent it back. Mobile operators generally liked these delivery mechanisms because they received a large portion of each messaging fee.

WAP fell short of commercial expectations. In some markets, such as Japan, it flourished, whereas in others, like the United States, it failed to take off. Handset screens were too small for surfing. Reading a sentence fragment at a time, and then waiting seconds for the next segment to download, ruined the user experience, especially because every second of downloading was often charged to the user. Critics began to call WAP "Wait and Pay."

Finally, the mobile operators who provided the WAP portal (the default home page loaded when you started your WAP browser) often restricted which WAP sites were accessible. The portal allowed the operator to restrict the number of sites users could browse and to funnel subscribers to the operator's preferred content providers and exclude competing sites. This kind of walled garden approach further discouraged third-party developers, who already faced difficulties in monetizing applications, from writing applications.

Proprietary Mobile Platforms

It came as no surprise when users wanted more—they will always want more.

Writing robust applications such as graphic-intensive video games with WAP was nearly impossible. The 18-year-old to 25-year-old sweet-spot demographic—the kids

with the disposable income most likely to personalize their phones with wallpapers and ringtones—looked at their portable gaming systems and asked for a device that was both a phone and a gaming device or a phone and a music player. They argued that if devices such as Nintendo's Game Boy could provide hours of entertainment with only five buttons, why not just add phone capabilities? Others looked to their digital cameras, Palms, Blackberries, iPods, and even their laptops and asked the same question. The market seemed to be teetering on the edge of device convergence.

Memory was getting cheaper; batteries were getting better; and PDAs and other embedded devices were beginning to run compact versions of common operating systems such as Linux and Windows. The traditional desktop application developer was suddenly a player in the embedded device market, especially with Smartphone technologies such as Windows Mobile, which they found familiar.

Handset manufacturers realized that if they wanted to continue to sell traditional handsets, they needed to change their protectionist policies pertaining to handset design and expose their internal frameworks, at least, to some extent.

A variety of different proprietary platforms emerged—and developers are still actively creating applications for them. Some Smartphone devices ran Palm OS (now Garnet OS) and RIM Blackberry OS. Sun Microsystems took its popular Java platform and J2ME emerged (now known as Java Micro Edition [Java ME]). Chipset maker Qualcomm developed and licensed its Binary Runtime Environment for Wireless (BREW). Other platforms, such as Symbian OS, were developed by handset manufacturers such as Nokia, Sony Ericsson, Motorola, and Samsung. The Apple iPhone OS (OS X iPhone) joined the ranks in 2008. Figure 1.4 shows several different phones, all of which have different development platforms.

Figure 1.4 Phones from various mobile device platforms.

Many of these platforms have associated developer programs. These programs keep the developer communities small, vetted, and under contractual agreements on what they can and cannot do. These programs are often required and developers must pay for them.

Each platform has benefits and drawbacks. Of course, developers love to debate over which platform is "the best." (Hint: It's usually the platform we're currently developing for.)

The truth is no one platform has emerged victorious. Some platforms are best suited for commercializing games and making millions—if your company has brand backing. Other platforms are more open and suitable for the hobbyist or vertical market applications. No mobile platform is best suited for all possible applications. As a result, the mobile phone has become increasingly fragmented, with all platforms sharing part of the pie.

For manufacturers and mobile operators, handset product lines became complicated fast. Platform market penetration varies greatly by region and user demographic. Instead of choosing just one platform, manufacturers and operators have been forced to sell phones for all the different platforms to compete. We've even seen some handsets supporting multiple platforms. (For instance, Symbian phones often also support J2ME.)

The mobile developer community has become as fragmented as the market. It's nearly impossible to keep track of all the changes in the market. Developer specialty niches have formed. The platform development requirements vary greatly. Mobile software developers work with distinctly different programming environments, different tools, and different programming languages. Porting among the platforms is often costly and not straightforward. Keeping track of handset configurations and testing requirements, signing and certification programs, carrier relationships, and application marketplaces have become complex spin-off businesses of their own.

It's a nightmare for the ACME Company wanting a mobile application. Should they develop a J2ME application? BREW? iPhone? Windows Mobile? Everyone has a different kind of phone. ACME is forced to choose one or, worse, all of the above. Some platforms allow for free applications, whereas others do not. Vertical market application opportunities are limited and expensive.

As a result, many wonderful applications have not reached their desired users, and many other great ideas have not been developed at all.

The Open Handset Alliance

Enter search advertising giant Google. Now a household name, Google has shown an interest in spreading its brand and suite of tools to the wireless marketplace. The company's business model has been amazingly successful on the Internet, and technically speaking, wireless isn't that different.

Google Goes Wireless

The company's initial forays into mobile were beset with all the problems you would expect. The freedoms Internet users enjoyed were not shared by mobile phone subscribers. Internet users can choose from the wide variety of computer brands, operating systems, Internet service providers, and Web browser applications.

Nearly all Google services are free and ad driven. Many applications in the Google Labs suite would directly compete with the applications available on mobile phones. The applications range from simple calendars and calculators to navigation with Google Maps and the latest tailored news from News Alerts—not to mention corporate acquisitions like Blogger and YouTube.

When this approach didn't yield the intended results, Google decided to a different approach—to revamp the entire system upon which wireless application development was based, hoping to provide a more open environment for users and developers: the Internet model. The Internet model allowes users to choose between freeware, shareware, and paid software. This enables free market competition among services.

Forming of the Open Handset Alliance

With its user-centric, democratic design philosophies, Google has led a movement to turn the existing closely guarded wireless market into one where phone users can move between carriers easily and have unfettered access to applications and services. With its vast resources, Google has taken a broad approach, examining the wireless infrastructure from the FCC wireless spectrum policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires.

Next, Google joined with other like-minded members in the wireless community and posed the following question: What would it take to build a better mobile phone?

The Open Handset Alliance (OHA) (Figure 1.5) was formed in November 2007 to answer that very question. The OHA is a business alliance comprised of many of the largest and most successful mobile companies on the planet. Its members include chip makers, handset manufacturers, software developers, and service providers. The entire mobile supply chain is well represented.

Figure 1.5 The Open Handset Alliance.

Working together, OHA members began developing a nonproprietary open standard platform that would aim to alleviate the aforementioned problems hindering the mobile community. They called it the Android project.

Google's involvement in the Android project has been extensive. The company hosts the open source project and provides online documentation, tools, forums, and the

Software Development Kit (SDK). Google has also hosted a number of events at conferences and the Android Developer Challenge, a contest to encourage developers to write killer Android applications—for $10 million dollars in prizes.

Manufacturers: Designing the Android Handsets

More than half the members of the OHA are handset manufacturers, such as Samsung, Motorola, HTC, and LG, and semiconductor companies, such as Intel, Texas Instruments, NVIDIA, and Qualcomm. These companies are helping design the first generation of Android handsets.

The first shipping Android handset—the T-Mobile G1—was developed by handset manufacturer HTC with service provided by T-Mobile. It was released in October 2008. Many other Android handsets are slated for 2009 and early 2010.

Content Providers: Developing Android Applications

When users have Android handsets, they need those killer apps, right?

Google has led the pack, developing Android applications, many of which, like the email client and Web browser, are core features of the platform. OHA members, such as eBay, are also working on Android application integration with their online auctions.

The first Android Developer Challenge received 1,788 submissions—all newly developed Android games, productivity helpers, and a slew of Location-Based Services (LBS). We also saw humanitarian, social networking, and mash-up apps. Many of these applications have debuted with users through the Android Market—Google's software distribution mechanism for Android.

Mobile Operators: Delivering the Android Experience

After you have the phones, you have to get them out to the users. Mobile operators from Asia, North America, Europe, and Latin America have joined the OHA, ensuring a market for the Android movement. With almost half a billion subscribers, telephony giant China Mobile is a founding member of the alliance. Other operators have signed on as well.

Taking Advantage of All Android Has to Offer

Android's open platform has been embraced by much of the mobile development community—extending far beyond the members of the OHA.

As Android phones and applications become more readily available, many in the tech community anticipate other mobile operators and handset manufacturers will jump on

the chance to sell Android phones to their subscribers, especially given the cost benefits compared to proprietary platforms. Already, North American operators, such as Verizon Wireless and AT&T, have shown an interest in Android, and T-Mobile already provides handsets.

If the open standard of the Android platform results in reduced operator costs in licensing and royalties, we could see a migration to open handsets from proprietary platforms such as BREW, Windows Mobile, and even the Apple iPhone. Android is well suited to fill this demand.

Android Platform Differences

Android is hailed as "the first complete, open, and free mobile platform."

- **Complete:** The designers took a comprehensive approach when they developed the Android platform. They began with a secure operating system and built a robust software framework on top that allows for rich application development opportunities.
- **Open:** The Android platform is provided through open source licensing. Developers have unprecedented access to the handset features when developing applications.
- **Free:** Android applications are free to develop. There are no licensing or royalty fees to develop on the platform. No required membership fees. No required testing fees. No required signing or certification fees. Android applications can be distributed and commercialized in a variety of ways.

Android: A Next Generation Platform

Although Android has many innovative features not available in existing mobile platforms, its designers also leveraged many tried-and-true approaches proven to work in the wireless world. It's true that many of these features appear in existing proprietary platforms, but Android combines them in a free and open fashion, while simultaneously addressing many of the flaws on these competing platforms.

The Android mascot is a little green robot, shown in Figure 1.6. You'll see this little guy (girl?) often used to depict Android-related materials.

Android is the first in a new generation of mobile platforms, giving its platform developers a distinct edge on the competition. Android's designers examined the benefits and drawbacks of existing platforms and then incorporate their most successful features. At the same time, Android's designers avoided the mistakes others suffered in the past.

Figure 1.6 The Android mascot.

Free and Open Source

Android is an open source platform. Neither developers nor handset manufacturers pay royalties or license fees to develop for the platform.

The underlying operating system of Android is licensed under GNU General Public License Version 2 (GPLv2), a strong "copyleft" license where any third-party improvements must continue to fall under the open source licensing agreement terms. The Android framework is distributed under the Apache Software License (ASL/Apache2), which allows for the distribution of both open and closed source derivations of the source code. Commercial developers (handset manufacturers especially) can choose to enhance the platform without having to provide their improvements to the open source community. Instead, developers can profit from enhancements such as handset-specific improvements and redistribute their work under whatever licensing they want.

Android application developers have the ability to distribute their applications under whatever licensing scheme they prefer. Developers can write open source freeware or traditional licensed applications for profit and everything in between.

Familiar and Inexpensive Development Tools

Unlike some proprietary platforms that require developer registration fees, vetting, and expensive compilers, there are no upfront costs to developing Android applications.

Freely Available Software Development Kit

The Android SDK and tools are freely available. Developers can download the Android SDK from the Android Web site after agreeing to the terms of the Android Software Development Kit License Agreement.

Familiar Language, Familiar Development Environments

Developers have several choices when it comes to integrated development environments (IDEs). Many developers choose the popular and freely available Eclipse IDE to design and develop Android applications. Eclipse is the most popular IDE for Android development and there is an Android plug-in available for facilitating Android development. Android applications can be developed on the following operating systems:

- Windows XP or Vista
- Mac OS X 10.4.8 or later (x86 only)
- Linux (tested on Linux Ubuntu 6.06 LTS, Dapper Drake)

Reasonable Learning Curve for Developers

Android applications are written in a well-respected programming language: Java.

The Android application framework includes traditional programming constructs, such as threads and processes and specially designed data structures to encapsulate objects commonly used in mobile applications. Developers can rely on familiar class libraries, such as `java.net` and `java.text`. Specialty libraries for tasks like graphics and database management are implemented using well-defined open standards like OpenGL Embedded Systems (OpenGL ES) or SQLite.

Enabling Development of Powerful Applications

In the past, handset manufacturers often established special relationships with trusted third-party software developers (OEM/ODM relationships). This elite group of software developers wrote native applications, such as messaging and Web browsers, which shipped on the handset as part of the phone's core feature set. To design these applications, the manufacturer would grant the developer privileged inside access and knowledge of a handset's internal software framework and firmware.

On the Android platform, there is no distinction between native and third-party applications, enabling healthy competition among application developers. All Android applications use the same libraries. Android applications have unprecedented access to the underlying hardware, allowing developers to write much more powerful applications. Applications can be extended or replaced altogether. For example, Android developers are now free to design email clients tailored to specific email servers such as Microsoft Exchange or Lotus Notes.

Rich, Secure Application Integration

If you recall the bat story I previously shared, you'll note that I accessed a wide variety of phone applications in the course of a few moments: text messaging, phone dialer, camera, email, picture messaging, and the browser. Each was a separate application running on the phone—some built-in and some purchased. Each had its own unique user interface. None were truly integrated.

Not so with Android. One of the Android platform's most compelling and innovative features is well-designed application integration. Android provides all the tools necessary to build a better "bat trap," if you will, by allowing developers to write applications that leverage core functionality such as Web browsing, mapping, contact management, and messaging seamlessly. Applications can also become content providers and share their data among each other in a secure fashion.

Platforms like Symbian have suffered from setbacks due to malware. Android's vigorous application security model helps protect the user and the system from malicious software.

No Costly Obstacles to Publication

Android applications have none of the costly and time-intensive testing and certification programs required by other platforms such as BREW and Symbian.

A "Free Market" for Applications

Android developers are free to choose any kind of revenue model they want. They can develop freeware, shareware, or trial-ware applications, ad-driven, and paid applications. Android was designed to fundamentally change the rules about what kind of wireless applications could be developed. In the past, developers faced many restrictions that had little to do with the application functionality or features:

- Store limitations on the number of competing applications of a given type
- Store limitations on pricing, revenue models, and royalties
- Operator unwillingness to provide applications for smaller demographics

With Android, developers can write and successfully publish any kind of application they want. Developers can tailor applications to small demographics, instead of just large-scale money-making ones often insisted upon by mobile operators. Vertical market applications can be deployed to specific, targeted users.

Because developers have a variety of application distribution mechanisms to choose from, they can pick the methods that work for them instead of being forced to play by others' rules. Android developers can distribute their applications to users in a variety of ways.

- Google developed the Android Market (Figure 1.7), a generic Android application store with a revenue-sharing model.
- Handango.com added Android applications to its existing catalogue using their billing models and revenue sharing model.
- Developers can come up with their own delivery and payment mechanisms.

Mobile operators are still free to develop their own application stores and enforce their own rules, but it will no longer be the only opportunity developers have to distribute their applications.

Figure 1.7 The Android market.

Android might be the next generation in mobile platforms, but the technology is still in its early stages. Early Android developers have had to deal with the typical roadblocks associated with a new platform: frequently revised SDKs, lack of good documentation, and market uncertainties. There are only a handful of Android handsets available to consumers at this time.

On the other hand, developers diving into Android development now benefit from the first-to-market competitive advantages we've seen on other platforms such as BREW and Symbian. Early developers who give feedback are more likely to have an impact on the long-term design of the Android platform and what features will come in the next version of the SDK. Finally, the Android forum community is lively and friendly. Incentive programs, such as the Android Developer Challenge, have encouraged many new developers to dig into the platform.

A New and Growing Platform

What's New in Android 1.5

The much-anticipated Android 1.5 SDK, released in late April 2009, provided a number of substantial improvements to both the underlying software libraries and the Android development tools and build environment. Also, the Android system received some much-needed UI "polish," both in terms of visual appeal and performance.

Although most of these upgrades and improvements were welcome and necessary, the new SDK version did cause some upheaval within the Android developer community. A number of published applications required retesting and resubmission to the Android Marketplace to conform to the new SDK requirements, which were quickly rolled out to all Android phones in the field as a firmware upgrade, rendering older applications obsolete.

The Android Platform

Android is an operating system and a software platform upon which applications are developed. A core set of applications for everyday tasks, such as Web browsing and email, are included on Android handsets.

As a product of the Open Handset Alliance's vision for a robust and open source development environment for wireless, Android is an emerging mobile development platform. The platform was designed for the sole purpose of encouraging a free and open market that all mobile applications phone users might want to have and software developers might want to develop.

Android's Underlying Architecture

The Android platform is designed to be more fault-tolerant than many of its predecessors. The handset runs a Linux operating system, upon which Android applications are executed in a secure fashion. Each Android application runs in its own virtual machine (Figure 1.8). Android applications are managed code; therefore, they are much less likely to cause the phone to crash, leading to fewer instances of device corruption (also called "bricking" the phone, or rendering it useless).

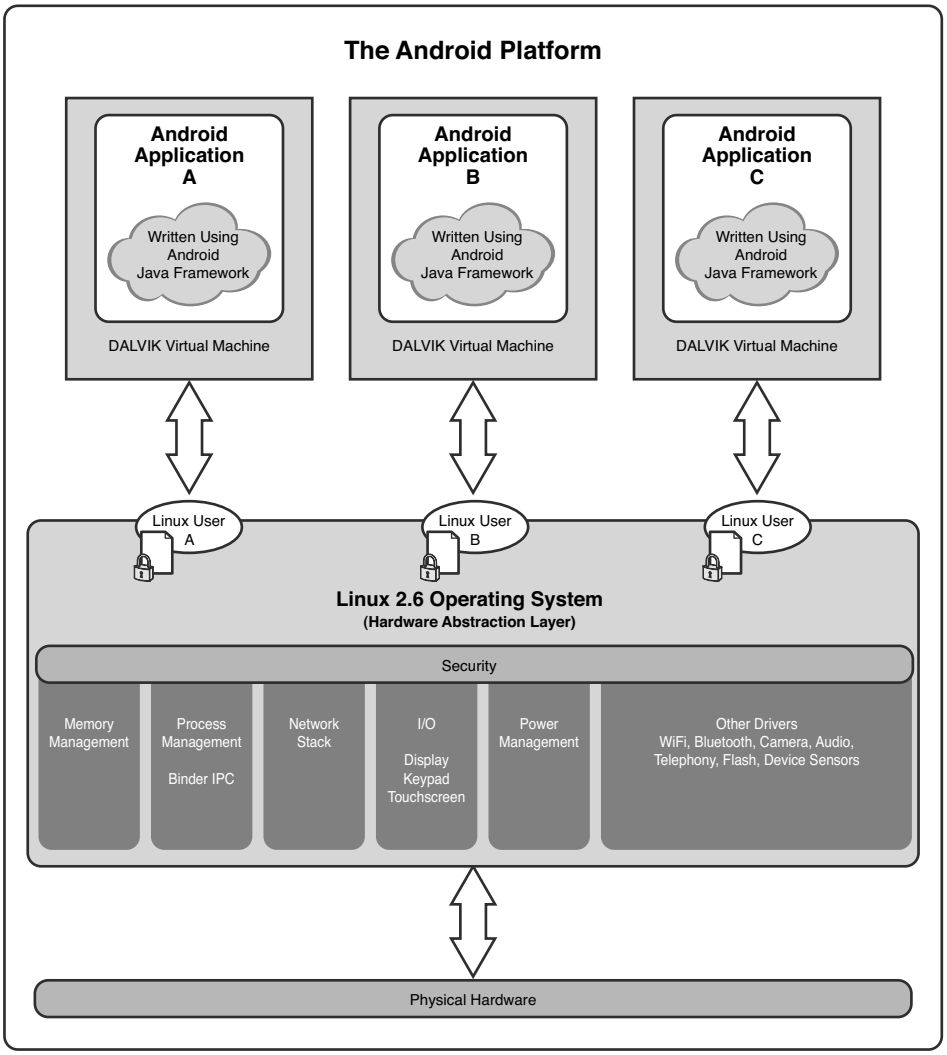

Figure 1.8 Diagram of the Android platform architecture.

The Linux Operating System

The Linux 2.6 kernel (Figure 1.9) handles core system services and acts as a hardware abstraction layer (HAL) between the physical hardware of the handset and the Android software stack.

Figure 1.9 Tux, the Linux kernel mascot.

What's New in Android 1.5

For Android 1.5, the Linux kernel received an upgrade from version 2.6.25 to 2.6.27. Although this type of change might not have an obvious effect for the typical Android developer, it is important to note that the kernel can and will be upgraded frequently. These seemingly minor incremental updates often include major security, performance, and functional features.

Kernel changes often have an impact on the security of the underlying device operating system and provide features and improvements for OEM-level Android device manufacturers. When stable, these features can be exposed to developers as part of an Android SDK upgrade, in the form of new APIs and performance enhancements to existing features.

The Android 1.5 version provides substantial feature enhancements, many of which tie back to features of the upgraded Linux kernel. Although the kernel memory footprint is larger, overall system performance has improved and a number of bugs have been fixed.

Some of the core functions the kernel handles include

- Enforcement of application permissions and security
- Low-level memory management
- Process management and threading
- The network stack
- Display, keypad input, camera, WiFi, Flash memory, audio, and binder (IPC) driver access

Android Application Runtime Environment

Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine (VM). Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint and multiple instances of the Dalvik VM can run concurrently on the handset.

Security and Permissions

The integrity of the Android platform is maintained through a variety of security measures.

Applications as Operating System Users

When an application is installed, the operating system creates a new user profile associated with the application. Each application runs as a different user, with its own private files on the file system, a user ID, and a secure operating environment.

The application executes in its own process with its own instance of the Dalvik VM and under its own user ID on the operating system.

Explicitly Defined Application Permissions

To access shared resources on the system, Android applications register for the specific privileges they require. Some of these privileges enable the application to use phone functionality to make calls, access the network, and control the camera and other hardware sensors. Applications also require permission to access shared data containing private and personal information such as user preferences, user's location, and contact information.

Applications might also enforce their own permissions by declaring them for other applications to use. The application can declare any number of different permission types, such as read-only or read-write permissions, for finer control over the application.

Limited Ad-Hoc Permissions

Applications that act as content providers might want to provide some on-the-fly permissions to other applications for specific information they want to share openly. This is done using ad-hoc granting and revoking of access to specific resources using Uniform Resource Identifiers (URIs).

URIs index specific data assets on the system, such as images and text. Here is an example of a URI that provides the phone numbers of all contacts:

```
content://contacts/phones
```

To understand how this permission process works, let's look at an example.

Let's say we've got an application that keeps track of the user's public and private birthday wish lists. If this application wanted to share its data with other applications, it could grant URI permissions for the public wish list, allowing another application permission to access this list without explicitly having to ask for it.

Application Signing for Trust Relationships

All Android applications packages are signed with a certificate, so users know that the application is authentic. The private key for the certificate is held by the developer. This helps establish a trust relationship between the developer and the user. It also allows the developer to control which applications can grant access to one another on the system. No certificate authority is necessary; self-signed certificates are acceptable.

Developing Android Applications

The Android SDK provides an extensive set of application programming interfaces (APIs) that is both modern and robust. Android handset core system services are exposed and accessible to all applications. When granted the appropriate permissions, Android applications can share data among one another and access shared resources on the system securely.

Android Programming Language Choices

Android applications are written in Java (Figure 1.10). For now, the Java language is the developer's only choice on the Android platform. There has been some speculation that other programming languages, such as C++, might be added in future versions of Android.

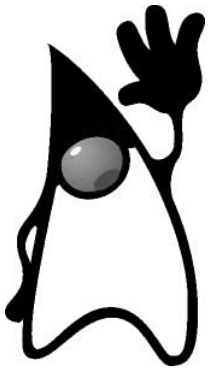

Figure 1.10 Duke, the Java mascot.

No Distinctions Made Between Native and Third-Party Applications

Unlike other mobile development platforms, there is no distinction between native applications and developer-created applications on the Android platform. Provided the application is granted the appropriate permissions, all applications have the same access to core libraries and the underlying hardware interfaces.

Android handsets ship with a set of native applications such as a Web browser and contact manager. Third-party applications might integrate with these core applications and even extend them to provide a rich user experience.

Commonly Used Packages

With Android, mobile developers no longer have to reinvent the wheel. Instead, developers use familiar class libraries exposed through Android's Java packages to perform common tasks such as graphics, database access, network access, secure communications, and utilities (such as XML parsing).

The Android packages include support for

- Common user interface widgets (Buttons, Spin Controls, Text Input)
- User interface layout
- Secure networking and Web browsing features (SSL, WebKit)
- Structured storage and relational databases (SQLite)
- Powerful 2D and 3D graphics (SGL and OpenGL ES 1.0)
- Audio and visual media formats (MPEG4, MP3, Still Images)
- Access to optional hardware such as Location-Based Services (LBS), WiFi, and Bluetooth

Android Application Framework

The Android application framework provides everything necessary to implement your average application. The Android application lifecycle involves the following key components:

- Activities are functions the application performs.
- Groups of views define the application's layout.
- Intents inform the system about an application's plans.
- Services allow for background processing without user interaction.
- Notifications alert the user when something interesting happens.

Android Applications can interact with the operating system and underlying hardware using a collection of managers. Each manager is responsible for keeping the state of some underlying system service. For example, there is a `LocationManager` that facilitates interaction with the location-based services available on the handset. The `ViewManager` and `WindowManager` manage user interface fundamentals.

Applications can interact with one another by using or acting as a `ContentProvider`. Built-in applications such as the Contact manager are content providers, allowing third-party applications to access contact data and use it in an infinite number of ways. The sky is the limit.

Summary

Mobile software development has evolved over time. Android has emerged as a new mobile development platform, building on past successes and avoiding past failures of other platforms. Android was designed to empower the developer to write innovative applications. The platform is open source, with no up-front fees, and developers enjoy many benefits over other competing platforms.

Now it's time to dive deeper and start writing Android code, so you can evaluate what Android can do for you. In the next chapter, we configure the Android development environment and take a brief walk through the Android SDK.

References and More Information

Android Development http://developer.android.com
Open Handset Alliance: http://www.openhandsetalliance.com