PAPER NO. CT 53

SECTION 5

CERTIFIED INFORMATION COMMUNICATION TECHNOLOGISTS (CICT)

MOBILE APPLICATION DEVELOPMENT

STUDY TEXT

15.0 LEARNING OUTCOMES

A candidate who passes this paper should be able to:
• Identify mobile applications, platforms and architecture
• Develop mobile applications Using development tools and strategies
• Test mobile applications
• Secure mobile applications

CONTENT

15.1 Mobile devices and applications

- Definition of mobile computing
- Types of mobile devices
- Uses of mobile devices
- Overview of mobile applications
- Mobile browsers

15.2 Introduction to mobile application development

- Mobile application challenges
- Mobile application development tools
- Mobile application programming languages
- Mobile application management
- Mobile application best practices

15.3 Mobile platforms and architectures

- Internet protocols for mobile applications
- Mobile application distribution platforms and environments
- Mobile application development architectures
- Styles of mobile architecture

15.4 Mobile application development

- Mobile application development lifecycle
- Functions. Arrays and objects
- Control structures and modes of execution
- Using HTML, CSS. DOM, JavaScript and JQuery

15.5 iOS application development

- Window-based application and MUC
- Objective-C programming
- User Interface Design
- Introduction to graphics on the iPhone
- Core data and localization
- Multi-threading and multi-tasking
- Web services and networking

15.6 Android application development

- Java reviews
- Androids SPK
- Resources, views and intents
- Intents and storage
- Storage and threads

15.7 Mobile application testing

- Merits and demerits of mobile application testing
- Challenges of mobile application testing
- Types of mobile application testing
- Testing tools

15.8 Mobile application security

- Reducing mobile risks
- Cloud based assessments and solutions
- Security strategies
- Security testing techniques and certification

15.9 Emerging issues and trends

CONTENT **PAGE**

CHAPTER 1

MOBILE DEVICES AND APPLICATIONS

- ***Definition of mobile computing***

Mobile computing is a generic term used to refer to a variety of devices that allow people to access data and information from where ever they are.
Also Known As: mobile device
Examples: Mobile computing can use cell phone connections to make phone calls as well as connecting to the Internet.

- ***Types of mobile devices***

We define mobile devices as having:

- the ability to connect to the Internet (or other data network)
- supports user input and interaction
- offers multiple functionalities
- is lightweight and is less than 10"

Devices that are "mobile devices":

- smartphone (and some feature phones)
- tablets
- netbooks and ultraportable laptop
- personal digital assistant (e.g. iPod Touch)
- GPS navigation device (a.k.a. car or personal navigation device)

Some participants used the "other" field to answer laptops and e-Readers. I also considered whether portable game consoles and digital audio guides (as some museums use) should be considered mobile devices.

Wikipedia's definition is pretty broad. To them a mobile is "small, hand-held computing device, typically having a display screen with touch input and/or a miniature keyboard and less than 2 pounds (0.91 kg)". Wikipedia lists calculators, digital cameras, and MP3 players as mobile device. I normally love Wikipedia but I think they are stretching the term to mean pretty much any portable electronic device.

Perhaps these are all just types of handheld computing devices. I think my definition fits the core functionality of what a device needs as a category term.

So although all the devices mentioned so far have some computing power and many have network connectivity (as even many e-Readers and digital cameras now have). But an e-Reader and digital camera are pretty much single function devices. The Kindle e-Reader does have the cool ability to of user interaction in the ability to highlight passages of eBooks and share them online with others, but users can't create substantial content and it essentially it is a single function device (hence the name even).
Laptops may be portable but they aren't portable enough to allow ubiquitous access - a trait that I think is central to the concept of mobile device (opposed to just portable device).

- ***Uses of mobile devices***

Handheld devices have become ruggedized for use in mobile field management. Uses include digitizing notes, sending and receiving invoices, asset management, recording signatures, managing parts, and scanning barcodes.

Recent developments in mobile collaboration systems employ handheld devices that combine video, audio and on-screen drawing capabilities to enable multi-party conferencing in real-time, independent of location.

Handheld computers are available in a variety of form factors, including smartphones on the low end, handheld PDAs, Ultra-Mobile PCs and Tablet PCs (Palm OS, WebOS).

Users can watch television through Internet on mobile devices. Mobile television receivers have existed since the 1960s, and in the 21st century mobile phone providers began making television available on cellular phones.

Nowadays, mobile devices can create, sync, and share everything we want despite of distance or specifications of mobile devices. In the medical field, mobile devices are quickly becoming essential tools for accessing clinical information such as drugs, treatment, and even medical calculation.

Due to the popularity of Candy Crush and other mobile device games, online casinos are also offering casino games on mobile devices. The casino games are available on iOS, Android, Windows Phone and Windows. Available games are roulette, blackjack and several different types of slots. Most casinos have a play for free option.

In the military field, mobile devices have created new opportunities for the Army to deliver training and educational materials to soldiers around the world.

- *Overview of mobile applications*

Mobile application development is a term used to denote the act or process by which application software is developed for handheld devices, such as personal digital assistants, enterprise digital assistants or mobile phones. These applications can be pre-installed on phones during manufacturing platforms, or delivered as web applications using server-side or client-side processing (e.g. JavaScript) to provide an "application-like" experience within a Web browser.

Application software developers also have to consider a lengthy array of screen sizes, hardware specifications and configurations because of intense competition in mobile software and changes within each of the platforms. Mobile app development has been steadily growing, both in terms of revenues and jobs created. A 2013 analyst report estimates there are 529,000 direct App Economy jobs within the EU 28 members, 60% of which are mobile app developers.

As part of the development process, Mobile User Interface (UI) Design is also an essential in the creation of mobile apps. Mobile UI considers constraints & contexts, screen, input and mobility as outlines for design. The user is often the focus of interaction with their device, and the interface entails components of both hardware and software. User input allows for the users to manipulate a system, and device's output allows the system to indicate the effects of the users' manipulation. Mobile UI design constraints include limited attention and form factors, such as a mobile device's screen size for a user's hand(s). Mobile UI contexts signal cues from user activity, such as location and scheduling that can be shown from user interactions within a mobile application.

Overall, mobile UI design's goal is primarily for an understandable, user-friendly interface. The UI of mobile apps should: consider users' limited attention, minimize keystrokes, and be task-oriented with a minimum set of functions. This functionality is supported by Mobile enterprise application platforms or Integrated development environments (IDEs).

CHAPTER 2

INTRODUCTION TO MOBILE APPLICATION DEVELOPMENT

Mobile application development is a term used to denote the act or process by which application software is developed for handheld devices, such as personal digital assistants, enterprise digital assistants or mobile phones. These applications can be pre-installed on phones during manufacturing platforms, or delivered as web applications using server-side or client-side processing (e.g. JavaScript) to provide an "application-like" experience within a Web browser. Application software developers also have to consider a lengthy array of screen sizes, hardware specifications and configurations because of intense competition in mobile software and changes within each of the platforms. Mobile app development has been steadily growing, both in terms of revenues and jobs created. A 2013 analyst report estimates there are 529,000 direct App Economy jobs within the EU 28 members, 60% of which are mobile app developers.

As part of the development process, Mobile User Interface (UI) Design is also an essential in the creation of mobile apps. Mobile UI considers constraints & contexts, screen, input and mobility as outlines for design. The user is often the focus of interaction with their device, and the interface entails components of both hardware and software. User input allows for the users to manipulate a system, and device's output allows the system to indicate the effects of the users' manipulation.

Mobile UI design constraints include limited attention and form factors, such as a mobile device's screen size for a user's hand(s). Mobile UI contexts signal cues from user activity, such as location and scheduling that can be shown from user interactions within a mobile application.

Overall, mobile UI design's goal is primarily for an understandable, user-friendly interface. The UI of mobile apps should: consider users' limited attention, minimize keystrokes, and be task-oriented with a minimum set of functions. This functionality is supported by Mobile enterprise application platforms or Integrated development environments (IDEs).

Mobile UIs, or front-ends, rely on mobile back-ends to support access to enterprise systems. The mobile back-end facilitates data routing, security, authentication, authorization, working off-line, and service orchestration. This functionality is supported by a mix of middleware components including mobile app servers, Mobile Backend as a service (MBaaS), and SOA infrastructure.

- ***Mobile application challenges***

Over the next few years, improving the convenience of mobile services will depend on improving the use of context in delivering mobile experiences. Your business will need to

predict what your customers want when they launch a mobile application or website. Delta Airlines, for example, knows how close a passenger is to departure time and delivers relevant content, such as a frequent flier's real-time status on the upgrade list for her next flight. Today, that kind of context is uncommon. Tomorrow, it will be table stakes.

Application developers writing mobile apps will have to start thinking about "mobile context" which we define as everything your customer has told you and all you can understand about what the customer is currently experiencing. Context is just one of the big, new challenges that application developers will face. Here are several more of the most important challenges we see.

1. Context.

Your customer's mobile context consists of:

Preferences: The history and personal decisions the customer has shared with you or with social networks.

Situation: The current location, of course, but other relevant factors could include the altitude, environmental conditions and even speed the customer is experiencing.

Attitude: The feelings or emotions implied by the customer's actions and logistics. Delivering a good contextual experience will require aggregating information from many sources. It could be from the devices customers are carrying, the local context of devices and sensors around them (e.g. a geofence that knows which airport gate they're at), an extended network of things they care about (e.g. the maintenance status of the incoming airplane they are about to take for their next flight, and the probability it will leave on time) and the historical context of their preferences. Gathering this data is a major challenge because it will be stored on multiple systems of record to which your app will need to connect.

2. Device Proliferation.

Another challenge facing mobile developers is device proliferation. It might seem like today's mobile app development process is pretty well defined: Build your app, make sure it looks pretty on a 4-inch smartphone and a 10-inch tablet, and then submit it to an app store. It's not quite that easy now, and it'll be much tougher in the near future. A wide range of new device sizes and changes to the nature of the apps themselves will increase the need for flexibility, especially on the client. We're already seeing 5-inch phablets, 7-inch tablets, and Windows 8 devices of 20 inches or more. Collectively, these new devices will significantly expand the potential for collecting contextual data about your customers. Here are some ideas of what changes you'll face:

3. Voice, Prioritized Over Touch.

Mobile developers are clamoring for API access to Apple's Siri and Google Now. There are a lot of scenarios where you would want to build voice input into your app today. For a running or fitness app, a phone is likely to be strapped to a person's sweaty arm, and looking at your screen while running can be a fast track into a lamp post. The same is true while driving. If you're

hustling through an airport with luggage to catch a flight, voice beats touch. Modern applications will let people use their devices while keeping their eyes and hands off it.

4. Heads-Up Interfaces.

Expect to see heads-up displays such as Google Glass go mainstream in the next five years as Moore's law pushes processors to the point where such gadgets can be made powerful, lightweight and perhaps even stylish. Augmented-reality apps that don't work well on a phone or tablet could be transformative when ported to a device like Google Glass. A compelling example would be an app that provides real-time information about the people you are talking to but whose names you've forgotten.

But heads-up displays will create a whole new slate of problems for developers. We'll have to adapt to peripheral cues such as reminders and alerts that don't block the user's vision. We'll also need to integrate tactile and aural feedback such as voice commands and vibrating sensors that alert users they need to take action.

5. Bigger -- And Smaller -- Touch Devices, And Adaptive UIs.

Today, most app developers prioritize a few popular devices, such as the iPhone, the Samsung Galaxy S III and the iPad. But cherry picking the most popular devices will become more of a challenge as device types and platforms proliferate. Google and Apple already support tablets of different sizes and, with Windows 8 now shipping, developers can expect to find a whole range of larger touch-sensitive devices, such as Hewlett-Packard's Envy series. But device surfaces will grow beyond specialized devices as the cost of multi-touch monitors falls -- to the point where touchwall computing becomes broadly available. Developers will need to scale their user interfaces, because an 84-inch experience is very different from a 4-inch experience.

6. Mobile Apps Become Pluggable Mobile Services.

Platform vendors such as Apple and Google are offering more platform-specific services that developers can leverage. Apple Passbook and Google Wallet are already established, but other examples such as Microsoft's Windows Phone 8 hubs and the Blackberry 10 "Peek" and "Flow" user interface further erode the distinction between a mobile platform and the apps that run on it. Over the next few years it will become more difficult to tell where the mobile platform services end and the third-party app begins.

CHAPTER 3

MOBILE PLATFORMS AND ARCHITECTURES

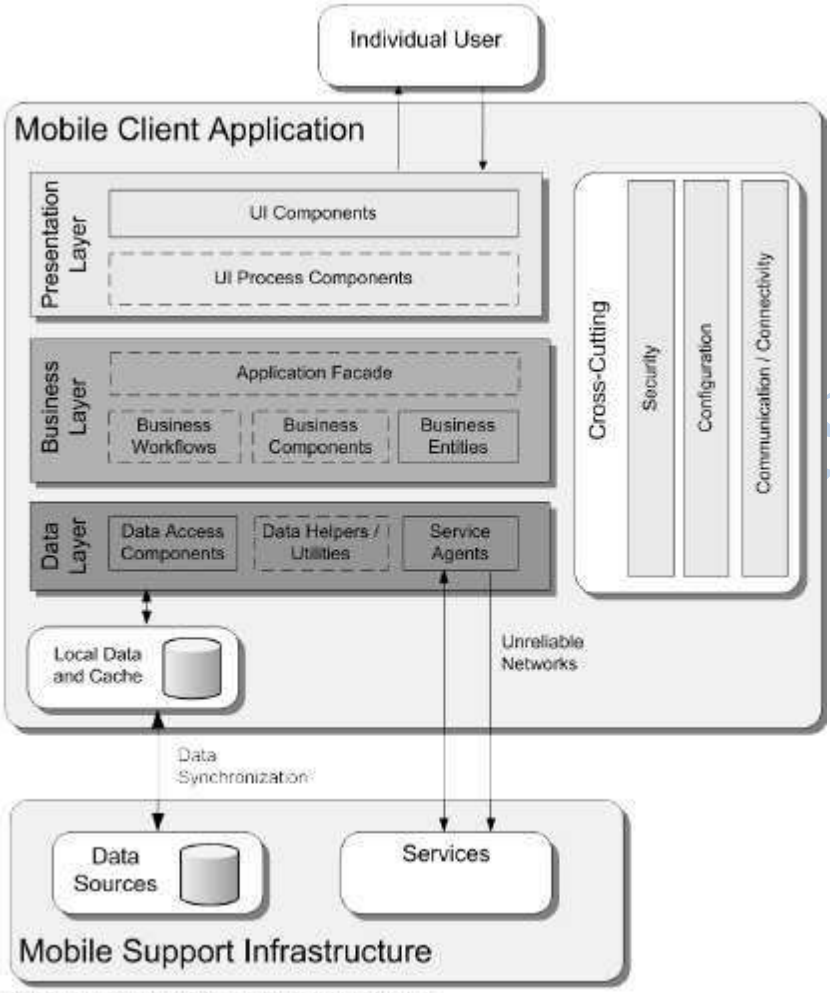


Figure 1 Common mobile application architecture

Architecture Frame

The following table lists the key areas to consider as you develop your architecture. Refer to the key issues in the table to understand where mistakes are most often made. The sections following this table provide guidelines for each of these areas.

Table 1 Architecture Frame

Area	Key issues
Authentication	• Lack of authorization across trust boundaries • Granular or improper authorization
Caching	• Caching volatile data not needed in an offline scenario • Caching sensitive data • Incorrect choice of caching store
Communication	• Incorrect choice of transport protocol • Chatty communication across physical and process boundaries • Failure to protect sensitive data
Concurrency and Transactions	• Not protecting concurrent access to static data • Deadlocks caused by improper locking • Not choosing the correct data concurrency model • Long-running transactions that hold locks on data • Using exclusive locks when not required

Area	Key issues
Configuration Management	• Lack of or incorrect configuration information • Not securing sensitive configuration information • Unrestricted access to configuration information
Coupling and Cohesion	• Incorrect grouping of functionality • No clear separation of concerns • Tight coupling across layers
Data Access	• Chatty calls to the database • Business logic mixed with data access code
Exception Management	• Failing to an unstable state • Revealing sensitive information to the end user • Using exceptions to control application flow • Not logging sufficient details about the exception
Layering	• Incorrect grouping of components within a layer • Not following layering and dependency rules
Logging and Instrumentation	• Lack of logging and instrumentation • Logging and instrumentation that is too fine-grained • Not making logging and instrumentation an option that is configurable at run time • Not suppressing and handling logging failures • Not logging business-critical functionality
State Management	• Using an incorrect state store • Not considering serialization requirements • Not persisting state when required
User Experience	• Not following published guidelines • Not considering accessibility

User Experience	• Not following published guidelines • Not considering accessibility • Creating overloaded interfaces with unrelated functionality
Validation	• Lack of validation across trust boundaries • Failure to validate for range, type, format, and length • Not reusing validation logic
Workflow	• Not considering management requirements • Choosing an incorrect workflow pattern • Not considering exception states and how to handle them

- ***Internet protocols for mobile applications***

Mobile IP is an Internet Engineering Task Force (IETF) standard communications protocol that is designed to allow mobile device users to move from one network to another while maintaining their permanent IP address. Defined in Request for Comments (RFC) 2002, Mobile IP is an enhancement of the Internet Protocol (IP) that adds mechanisms for forwarding Internet traffic to mobile devices (known as mobile nodes) when they are connecting through to other than their home network.

In traditional IP routing, IP addresses represent a topology. Routing mechanisms rely on the assumption that each network node will always have the same point of attachment to the Internet, and that each node's IP address identifies the network link where it is connected. Core Internet routers look at the IP address prefix, which identifies a device's network.
At the network level, routers look at the next few bits to identify the appropriate subnet. Finally, at the subnet level, routers look at the bits identifying a particular device. In this routing scheme, if you disconnect a mobile device from the Internet and want to reconnect through a different subnet, you have to configure the device with a new IP address, and the appropriate netmask and default router. Otherwise, routing protocols have no means of delivering packets because the device's IP address doesn't contain the necessary information about the current point of attachment to the Internet.

CHAPTER 4

MOBILE APPLICATION DEVELOPMENT

- ***Mobile application development lifecycle***

Introduction to the Mobile Software Development Lifecycle
Considerations when developing mobile applications PDF for offline use:

- Introduction to Mobile Development
- Hello, iOS
- Hello, Android
- Application Fundamentals

This article discusses the software development lifecycle with respect to mobile applications, and discusses some of the considerations required when building mobile projects. For developers wishing to just jump right in and start building, this guide can be skipped and read later for a more complete understanding of mobile development.

Overview
Building mobile applications can be as easy as opening up your IDE, throwing something together, doing a quick bit of testing, and submitting to an App Store – all done in an afternoon. Or it can be an extremely involved process that involves rigorous up-front design, usability testing, QA testing on thousands of devices, a full beta lifecycle, and then deployment a number of different ways.

In this document, we're going to take a thorough introductory examination of building mobile applications, including:

1. **Process** – The process of software development is called the Software Development Lifecycle (SDLC). We'll examine all phases of the SDLC with respect to mobile application development, including: **Inspiration, Design, Development, Stabilization, Deployment, and Maintenance.**
2. **Considerations** – There are a number of considerations when building mobile applications, especially in contrast to traditional web or desktop applications. We'll examine these considerations and how they affect mobile development.

This document is intended to answer fundamental questions about mobile app development, for new and experienced application developers alike. It takes a fairly comprehensive approach to introducing most of the concepts you'll run into during the entire Software Development Lifecycle (SDLC). However, this document may not be for everyone, if you're itching to just start building applications, we recommend jumping ahead to either the Introduction to Mobile

Development, Hello, Android or Hello, iPhone tutorials, and then coming back to this document later.

Mobile Development SDLC

The lifecycle of mobile development is largely no different than the SDLC for web or desktop applications. As with those, there are usually 5 major portions of the process:

1. **Inception** – All apps start with an idea. That idea is usually refined into a solid basis for an application.
2. **Design** – The design phase consists of defining the app's User Experience (UX) such as what the general layout is, how it works, etc., as well as turning that UX into a proper User Interface (UI) design, usually with the help of a graphic designer.
3. **Development** – Usually the most resource intensive phase, this is the actual building of the application.
4. **Stabilization** – When development is far enough along, QA usually begins to test the application and bugs are fixed. Often times an application will go into a limited beta phase in which a wider user audience is given a chance to use it and provide feedback and inform changes.
5. **Deployment**

   Often many of these pieces are overlapped, for example, it's common for development to be going on while the UI is being finalized, and it may even inform the UI design. Additionally, an application may be going into a stabilization phase at the same that new features are being added to a new version.

Furthermore, these phases can be used in any number of SDLC methodologies such as Agile, Spiral, Waterfall, etc.

Let's cover how each of these phases plays a part in Mobile Development.

1. Inception

The ubiquity and level of interaction people have with mobile devices means that nearly everyone has an idea for a mobile app. Mobile devices open up a whole new way to interact with computing, the web, and even corporate infrastructure.

The inception stage is all about defining and refining the idea for an app. In order to create a successful app, it's important to ask some fundamental questions. For example, if you're developing an app for distribution in a public app store, some considerations are:

- **Competitive Advantage** – Are there similar apps out there already? If so, how does this application differentiate from others?

  If you're intending for the app to be distributed in the enterprise:
- **Infrastructure Integration** – What existing infrastructure will it integrate with or extend? Additionally, you should evaluate the usage of the app in a mobile form factor:
- **Value** – What value does this app bring users? How will they use it?

- **Form/Mobility** – How will this app work in a mobile form factor? How can I add value using mobile technologies such as location awareness, the camera, etc.?

To help with designing the functionality of an app, it can be useful to define Actors and Use Cases. Actors are roles within an application and are often users. Use cases are typically actions or intents.
For instance, if you're building a task tracking application, you might have two Actors: *User* and *Friend*. A User might *Create a Task*, and *Share a Task* with a Friend. In this case, creating a task and sharing a task are two distinct use cases that, in tandem with the Actors, will inform what screens you'll need to build, as well as what business entities and logic will need to be developed.
If you've captured the appropriate use cases and actors, it's much easier to begin designing an application because you know exactly what you need to design, so the question becomes, how to design it, rather than what to design.

2. Designing Mobile Applications

Once you have a good idea of what it is you want to design, the next step is start trying to solve the User Experience or UX.

User Experience (UX) Design

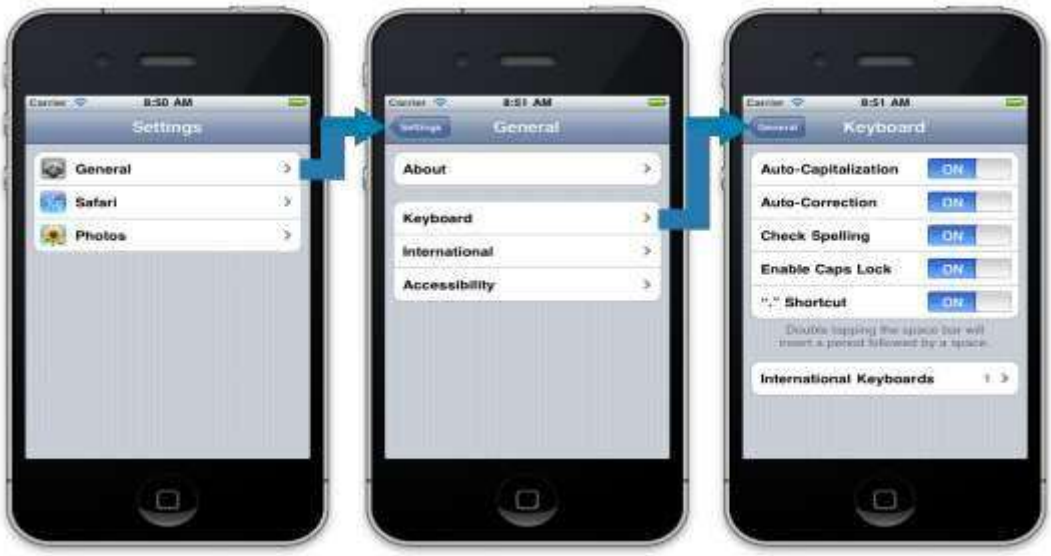
UX is usually done via wireframes or mockups using tools such as Balsamiq, Mockingbird, Visio, or just plain ol' pen and paper. UX Mockups allow you to quickly design UX without having to worry about the actual UI design:

When creating UX Mockups, it's important to consider the Interface Guidelines for the various platforms that you're designing for. By adhering to platform-specific guidelines, you can ensure that your apps feel at home on each platform. You can find each guide as follows:

1. **Apple** - Human Interface Guidelines
2. **Android** – Design Guidelines
3. **Windows Phone** – Design library for Windows Phone

For example, each app has a metaphor for switching between sections in an application. iOS uses a tabbar at the bottom of the screen, Android uses a tabbar at the top of the screen, and Windows Phone uses the Panorama view:

Additionally, the hardware itself also dictates UX decisions. For example, iOS devices have no physical *back* button, and



therefore introduce the Navigation Controller metaphor:

Furthermore, form factor also influences UX decisions. A tablet has far more real estate, so you can fit more information, and often what needs multiple screens on a phone is compressed into one for a tablet:

And due to the myriad of form factors out there, there are often mid-size form factors (somewhere between a phone and a tablet) that you may also want to target.

CHAPTER 5

IOS APPLICATION DEVELOPMENT

iOS is the operating system that runs on iPad, iPhone, and iPod touch devices. The operating system manages the device hardware and provides the technologies required to implement native apps. The operating system also ships with various system apps, such as Phone, Mail, and Safari, that provide standard system services to the user.

The *iOS Software Development Kit (SDK)* contains the tools and interfaces needed to develop, install, run, and test native apps that appear on an iOS device‘s Home screen. Native apps are built using the iOS system frameworks and Objective-C language and run directly on iOS. Unlike web apps, native apps are installed physically on a device and are therefore always available to the user, even when the device is in Airplane mode. They reside next to other system apps, and both the app and any user data is synced to the user‘s computer through iTunes.

At a Glance

The iOS SDK provides the resources you need to develop native iOS apps. Understanding a little about the technologies and tools contained in the SDK can help you make better choices about how to design and implement your apps.

The iOS Architecture Is Layered

At the highest level, iOS acts as an intermediary between the underlying hardware and the apps you create. Apps do not talk to the underlying hardware directly. Instead, they communicate with the hardware through a set of well-defined system interfaces. These interfaces make it easy to write apps that work consistently on devices having different hardware capabilities.

The implementation of iOS technologies can be viewed as a set of layers, which are shown in Figure I-1. Lower layers contain fundamental services and technologies. Higher-level layers build upon the lower layers and provide more sophisticated services and technologies.

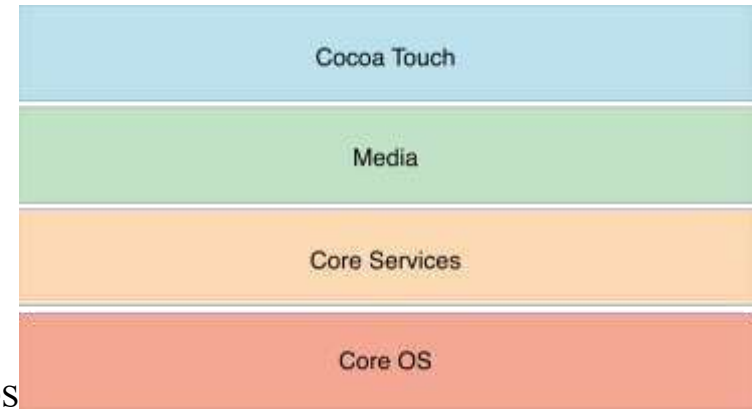


Figure I-1 Layers of iOS

As you write your code, it is recommended that you prefer the use of higher-level frameworks over lower-level frameworks whenever possible. The higher-level frameworks are there to provide object-oriented abstractions for lower-level constructs. These abstractions generally make it much easier to write code because they reduce the amount of code you have to write and encapsulate potentially complex features, such as sockets and threads. You may use lower-level frameworks and technologies, too, if they contain features not exposed by the higher-level frameworks.

- ***Window-based application and MUC***

Choosing a Programming Language for Windows Mobile Development

Windows Mobile 6.5
4/19/2010

There are several approaches which can be taken when developing applications for Windows Mobile devices. In this topic, we'll look at the various options and provide links to sources of more information.

Visual C++

Visual C++ is known as a "native" development language, as it talks directly to the hardware for the Windows Mobile device, with no intervening layers (unlike Visual C#, for example). Programming using C++ can be challenging, as it is not a trivial language to learn. Any errors in a C++ program, for example, accessing memory that has been freed, or forgetting to free memory, can potentially crash the entire device.

The advantages of using Visual C++ are execution speed, application size and flexibility. Applications written in C++ run very quickly and consume minimal resources: fast-action games

are good examples of programs that benefit from C++. Furthermore, the ability to access low-level system components means that using C++ is the only way to create a Today or Home screen plug-in.

A good way to learn Visual C++ is to investigate the free Visual Studio Visual C++ Express Edition, watch the video training and WebCasts, and read through the documentation. Although the Express Edition of Visual Studio does not allow you to develop applications for Windows Mobile, almost everything you will learn about application development can be applied directly to mobile devices.

Visual C++ applications can interact with the Windows Mobile device by calling the Win32 APIs (Application Program Interface functions). These APIs are functions that perform particular actions, such as playing a sound or drawing a button on the screen. There are thousands of these APIs (Windows Mobile supports a subset of the complete "desktop" Windows set of Win32 APIs), and they are documented in the section entitled Windows Mobile Features (Native). When browsing this section, pay particular attention to the fact that some APIs are available only in Windows Embedded CE - a platform that is related to, but separate from, Windows Mobile. A table in the top right of each topic will clarify which API is supported by which platform.

If you have experience developing for Windows using Visual C++, you will not find the transition to Windows Mobile particularly jarring. You should read the sections covering installing and using the tools, and then the topic Making use of Device-Specific Features which will highlight the unique abilities of Windows Mobile devices.

To begin a Visual C++ application, start Visual Studio, and select File, New, Project and select Smart Device under the Visual C++ node.

If you are new to both programming and Windows Mobile, it may be a good idea to begin with Visual C#, and then transition to Visual C++.

Visual C# and Visual Basic

Visual C# and Visual Basic .NET are "managed" development languages. Not only are they relatively easy to learn, but they support the .NET Compact Framework - a library of classes that perform a lot of frequently used programming tasks, to greatly simplify application development.

The development tools for C# and Visual Basic .NET include a fully what-you-see-is-what-you-get user interface designer. You can drag and drop buttons and other controls directly onto your application's window (called a "form" in managed programming), and then double-click to access the underlying code. This approach makes creating an application's user interface extremely fast and easy.

Extra classes covering everything from data structures to intercepting text messages are available to you as part of the Compact Framework library. You can read more about the framework in the section entitled the .NET Compact Framework Reference. To make use of Windows Mobile specific features, a set of extra classes are provided. These provide access to the device's features, for example, the list of Contacts, or built-in camera. The documentation for these classes is in the section entitled Windows Mobile Features (Managed).

If you have experience developing applications for Windows using Visual C#, the transition should be relatively painless. The Compact Framework is a subset of the .NET Framework, so some functionality may require a slight reworking of your code.

Visual C# is a great way to learn programming. You can learn more about using Visual C# on MSDN: for example, here is a topic entitled the Visual C# Programming Guide. To learn more about Visual Basic, here is another topic on MSDN: Getting Started with Visual Basic.

For more information, see the topic Developing with Managed Code.

To begin a Visual C# or Visual Basic .NET application, start Visual Studio, and select File, New, Project and select Smart Device under the relevant language node.

Client-side JScript

The web browser included with Windows Mobile devices - Internet Explorer Mobile - supports JScript. JScript is a superset of the language most commonly known as JavaScript. JScript programs are plain text files that are executed by the web browser. They can be embedded in an HTML page, or stored in separate files.

A JScript application is executed inside the web browser, and uses the web browser's window for input and output. It is possible to make use of AJAX programming techniques to provide a degree of user interaction, and to communicate with a remote server. Due to the nature of JScript, applications cannot access local data other than through cookies, which will introduce some limitations.

No developer tools other than a text editor are required to create a JScript application. The program may be stored locally, or accessed from a Web Server. For more information, see the section Programming with Internet Explorer Mobile and AJAX.

ASP.NET

While JScript is a client-side solution to writing Internet-style applications, ASP.NET is a server-side solution. With ASP.NET, you can write applications in C# or Visual Basic .NET that reside

on a Web Server, and perform complex processing, including creating user interface controls, and accessing databases. ASP.NET isolates the device characteristics from the application, making it straightforward to run one application on many difference device-types.

For an introduction to developing for mobile devices using ASP.NET, see Creating ASP.NET Mobile Web Pages.

- ***Objective-C programming***

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. This is the main programming language used by Apple for the OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch.
This reference will take you through simple and practical approach while learning Objective-C Programming language.

Audience
This reference has been prepared for the beginners to help them understand the basic to advanced concepts related to Objective-C Programming languages.

Prerequisites
Before you start doing practice with various types of examples given in this reference, I'm making an assumption that you are already aware about what is a computer program and what is a computer programming language?

Compile/Execute Objective-C Programs
For most of the examples given in this tutorial, you will find **Try it** option to compile and execute Objective-C programs online, so just make use of it and enjoy your learning.
Try the following example using **Try it** option available at the top right corner of the below sample code box −

```
#import <Foundation/Foundation.h>
int main()
{
    /* my first program in Objective-C */
     NSLog(@"Hello, World! \n");

     return 0;
    }
```

What is Objective-C?

If you're reading this series then I'll hazard a guess that you already know, but for those of you who don't, don't worry as by the end of this part you'll know what it is back-to-front and inside-out.

Objective-C is an object oriented language which lies on top of the C language (but I bet you guessed that part!). It's primary use in modern computing is on Mac OS X as a desktop language and also on iPhone OS (or as it is now called: iOS). It was originally the main language for NeXTSTEP OS, also known as the operating system Apple bought and descended Mac OS X from, which explains why its primary home today lies on Apple's operating systems.

Because Objective-C is a strict superset of C, we are free to use C in an Objective-C file and it will compile fine. Because any compiler of Objective-C will also compile any straight C code passed into it, we have all the power of C along with the power of objects provided by Objective-C.

If you're a little confused at this point, think of it this way: everything C can do, Objective-C can do too, but not the other way around.

What will I need?

Throughout this series, we will not focus on building applications for the iPhone. Instead, we will concentrate more on the language itself and for this reason all you will need is a Mac with a compiler such as GCC. If you've installed the developer tools from Apple (Xcode, Interface Builder, etc), then GCC should already be installed. If not, then head to Apple's developer website and get yourself a free copy.

CHAPTER 6

ANDROID APPLICATION DEVELOPMENT

- ***Java reviews***

Java for Mobile Devices is a set of technologies that let developers deliver applications and services to all types of mobile handsets, ranging from price efficient feature-phones to the latest smartphones. Java is currently running on over 3 billion phones worldwide, and growing. It offers unrivaled potential for the distribution and monetization of mobile applications.

At the core of the Java Mobile Platform is Java Platform, Micro Edition (Java ME). Java ME provides a robust, flexible environment for applications running on mobile and other embedded devices: mobile phones, TV set-top boxes, e-readers, Blu-Ray readers, printers and more. For over a decade, Oracle has been working along with leading mobile and embedded companies to develop the Java ME Platform through the Java Community Process (JCP). A key achievement has been the definition of the Mobile Services Architecture (MSA), setting a baseline of mobile APIs that developer can target within their applications. In 2011, Oracle and partners will be working within JCP to drive Java ME.next - a proposal for the modernization of Java ME .

In addition to its role within JCP, Oracle is also a provider of high performance Java ME implementations and developer technologies being used to deploy tens of thousands of applications worldwidein the mobile and embedded markets, including:

- Oracle Java Wireless Client: a multitasking Java ME runtime optimized for the leading mobile phone platforms.
- Java ME SDK: a state-of-the-art toolbox for developing and testing mobile applications.
- Light Weight UI Toolkit (LWUIT): a compact library for the creation of rich user interfaces.
- Oracle Java ME Embedded: designed and optimized to meet the unique requirements of small, low power devices.

- ***Androids SPK***

Android application package (**APK**) is the packagefile format used by the Android operating system for distribution and installation of application software and middleware.

APK files are analogous to other software packages such as MSI packages in Microsoft Windows or Deb packages in Debian-based operating systems like Ubuntu. To make an APK file, a program for Android is first compiled, and then all of its parts are packaged into one file. An APK file contains all of that program's code (such as .dex files), resources, assets, certificates, and manifest file. As is the case with many file formats, APK files can have any name needed, provided that the file name ends in ".apk".

APK files are a type of archive file, specifically in zip format packages based on the JAR file format, with `.apk` as the filename extension. The MIME type associated with APK files is `application/vnd.android.package-archive`.

APK file can be installed on Android powered devices just like installing software on PC. To secure the device, there is an "Unknown Sources" setting in Settings menu which is disabled by default. It must be enabled before installing any application with APK file. Enabling this setting is not required when you are installing anything via Google Play.

Package contents

An APK file is an archive that usually contains the following files and directories:

- `META-INF` directory:
  - `MANIFEST.MF`: the Manifest file
  - `CERT.RSA`: The certificate of the application.
  - `CERT.SF`: The list of resources and SHA-1 digest of the corresponding lines in the MANIFEST.MF file; for example:

```
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: wxqnEAI0UA5nO5QJ8CGMwjkGGWE=
...
Name: res/layout/exchange_component_back_bottom.xml
SHA1-Digest: eACjMjESj7Zkf0cBFTZ0nqWrt7w=
...
Name: res/drawable-hdpi/icon.png
SHA1-Digest: DGEqylP8W0n0iV/ZzBx3MW0WGCA=
```

- `lib`: the directory containing the compiled code that is specific to a software layer of a processor, the directory is split into more directories within it:
  - `armeabi`: compiled code for all ARM based processors only
  - `armeabi-v7a`: compiled code for all ARMv7 and above based processors only
  - `arm64-v8a`: compiled code for all ARMv8 arm64 and above based processors only
  - `x86`: compiled code for x86 processors only

- `x86_64`: compiled code for x86_64 processors only
- `mips`: compiled code for MIPS processors only

- `res`: the directory containing resources not compiled into resources.arsc (see below).
- `assets`: a directory containing applications assets, which can be retrieved by `AssetManager`.
- `AndroidManifest.xml`: An additional Android manifest file, describing the name, version, access rights, referenced library files for the application. This file may be in Android binary XML that can be converted into human-readable plaintext XML with tools such as AXMLPrinter2, android-apktool, or Androguard.
- `classes.dex`: The classes compiled in the dex file format understandable by the Dalvik virtual machine
- `resources.arsc`: a file containing precompiled resources, such as binary XML for example.

Android software development

Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in Java programming language using the Android software development kit (SDK), but other development environments are also available.

As of July 2013, more than one million applications have been developed for Android, with over 25 billion downloads. A June 2011 research indicated that over 67% of mobile developers used the platform, at the time of publication. In Q2 2012, around 105 million units of Android smartphones were shipped which acquires a total share of 68% in overall smartphones sale till Q2 2012

Official development tools

Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, and Windows XP or later. As of March 2015, the SDK is not available on Android itself, but the software development is possible by using specialized Android applications.Until around the end of 2014, the officially supported integrated development environment (IDE) was Eclipse using the Android Development Tools (ADT) Plugin, though IntelliJ IDEA IDE (all editions) fully supports Android development out of the box, and NetBeans IDE also supports Android development via a plugin. As of 2015, Android Studio, made by Google and powered by IntelliJ,

is the official IDE; however, developers are free to use others. Additionally, developers may use any text editor to edit Java and XML files, then use command line tools (Java Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in .apk format and stored under `/data/app` folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called Dalvik executables), resource files, etc.

Android Debug Bridge

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package. It consists of both client and server-side programs that communicate with one another. The ADB is typically accessed through the command-line interface, although numerous graphical user interfaces exist to control ADB.

The format for issuing commands through the ADB is typically:

```
adb [-d|-e|-s <serialNumber>] <command>
```

In a security issue reported in March 2011, ADB was targeted as a vector to attempt to install a rootkit on connected phones using a "resource exhaustion attack".

CHAPTER 7

MOBILE APPLICATION TESTING

Mobile application testing is a process by which application software developed for hand held mobile devices is tested for its functionality, usability and consistency. Mobile application testing can be automated or manual type of testing. Mobile applications either come pre-installed or can be installed from mobile software distribution platforms. Mobile devices have witnessed a phenomenal growth in the past few years. A study conducted by the Yankee Group predicts the generation of $4.2 billion in revenue by 2013 through 7 billion U.S. smartphone app downloads

MORE NOTES:

Mobile application testing

Mobile applications are first tested within the development environment using emulators and later subjected to field testing. Emulators provide an inexpensive way to test applications on mobile phones to which developers may not have physical access. The following are examples of tools used for testing application across the most popular mobile operating systems.

- **Google Android Emulator** - Google Android Emulator is an Android emulator that is patched to run on a Windows PC as a standalone app, without having to download and install the complete and complex Android SDK. It can be installed and Android compatible apps can be tested on it.
- **The official Android SDK Emulator** - The official Android SDK Emulator includes a mobile device emulator which mimics all of the hardware and software features of a typical mobile device (without the calls).
- **MobiOne** - MobiOne Developer is a mobile **WebIDE** for Windows that helps developers to code, test, debug, package and deploy mobile Web applications to devices such as iPhone, BlackBerry, Android, and the Palm Pre.

TestiPhone - TestiPhone is a web browser-based simulator for quickly testing iPhoneweb applications. This tool has been tested and works using Internet Explorer 7, Firefox 2 and Safari 3.

CHAPTER 8

MOBILE APPLICATION SECURITY

1. Mobile Devices Need Antimalware Software

A quick look at new malware threats discovered in the wild shows that mobile operating systems such as iOS and (especially) Android are increasingly becoming targets for malware, just as Windows, MacOS, and Linux have been for years. Anybody who wants to use a mobile device to access the Internet should install and update antimalware software for his or her smartphone or tablet. This goes double for anyone who wants to use such a device for work.

2. Secure Mobile Communications

Most experts recommend that all mobile device communications be encrypted as a matter of course, simply because wireless communications are so easy to intercept and snoop on. Those same experts go one step further to recommend that any communications between a mobile device and a company or cloud-based system or service require use of a VPN for access to be allowed to occur. VPNs not only include strong encryption, they also provide opportunities for logging, management and strong authentication of users who wish to use a mobile device to access applications, services or remote desktops or systems.

3. Require Strong Authentication, Use Password Controls

Many modern mobile devices include local security options such as built-in biometrics — fingerprint scanners, facial recognition, voiceprint recognition and so forth — but even older devices will work with small, portable security tokens (or one-time passwords issued through a variety of means such as email and automated phone systems). Beyond a simple account and password, mobile devices should be used with multiple forms of authentication to make sure that possession of a mobile device doesn't automatically grant access to important information and systems.

Likewise, users should be instructed to enable and use passwords to access their mobile devices. Companies or organizations should consider whether the danger of loss and exposure means that some number of failed login attempts should cause the device to wipe its internal storage clean. (Most modern systems include an ability to remotely wipe a smartphone or tablet, but mobile device management systems can bring that capability to older devices as well.)

4. Control Third-party Software

Companies or organizations that issue mobile devices to employees should establish policies to limit or block the use of third-party software. This is the best way to prevent possible compromise and security breaches resulting from intentional or drive-by installation of rogue software, replete with backdoors and "black gateways" to siphon information into the wrong hands.

For BYOD management, the safest course is to require such users to log into a remote virtual work environment. Then, the only information that goes to the mobile device is the screen output from work applications and systems; data therefore doesn't persist once the remote session ends. Since remote access invariably occurs through VPN connections, communications are secure as well — and companies can (and should) implement security policies that prevent download of files to mobile devices.

5. Create Separate, Secured Mobile Gateways

It's important to understand what kinds of uses, systems and applications mobile users really need to access. Directing mobile traffic through special gateways with customized firewalls and security controls in place — such as protocol and content filtering and data loss prevention tools — keeps mobile workers focused on what they can and should be doing away from the office. This also adds protection to other, more valuable assets they don't need to access on a mobile device anyway.

6. Choose (or Require) Secure Mobile Devices, Help Users Lock Them Down

Mobile devices should be configured to avoid unsecured wireless networks, and Bluetooth should be hidden from discovery. In fact, when not in active use for headsets and headphones, Bluetooth should be disabled altogether. Prepare a recommended configuration for personal mobile devices used for work — and implement such configurations *before* the intended users get to work on their devices.

7. Perform Regular Mobile Security Audits, Penetration Testing

At least once a year, companies and organizations should hire a reputable security testing firm to audit their mobile security and conduct penetration testing on the mobile devices they use. Such firms can also help with remediation and mitigation of any issues they discover, as will sometimes be the case. Hire the pros to do unto your mobile devices what the bad guys will try to do unto you sooner or later, though, and you'll be able to protect yourself from the kinds of threats they can present.

Security, Mobile or Otherwise, Is a State of Mind

While mobile security may have its own special issues and challenges, it's all part of the security infrastructure you must put in place to protect your employees, your assets and, ultimately, your reputation and business mission. By taking appropriate steps to safeguard against loss and mitigate risks, your employees and contractors will be able to take advantage of the incredible benefits that mobile devices can bring to the workplace.

Just remember the old adage about an ounce of prevention. That way, you're not saddled with costs or slapped with legal liabilities or penalties for failing to exercise proper prudence, compliance and best practices.

- ***Reducing mobile risks***

Risky business:

When a mobile device is lost or stolen, any business data it contains is jeopardized. Laws, such as California SB1386 (and similar laws introduced in 35 states last year), require companies to notify individuals whose private information may have been compromised. And businesses that violate industry mandates like HIPAA and GLBA face hefty fines or even jail time. But many companies cannot even enumerate the data carried by lost or stolen mobile devices.

A growing number of workers are using PDAs and smartphones to access business networks and applications. In the Nokia study, commonly-used mobile applications included e-mail, instant messaging, corporate database access, sales force automation, field service, CRM and ERP/supply chain applications. Companies without mobile-specific applications may still face mobile exposure through traditional applications. For example, many employees synchronize company e-mail onto PDAs or forward messages to smartphones. Therefore, if lost or stolen, these devices can be used to gain unauthorized access to an otherwise private network and applications therein.

Additionally, many mobile devices now support multiple wireless interfaces, creating new attack vectors. Mobile phones with Bluetooth can be "BlueBugged" (used by an attacker to place calls) or "BlueSnarfed" (accessed to retrieve contacts and calendars). Cradled PDAs can become Wi-Fi bridges into corporate networks. When used correctly, wireless interfaces can aid productivity, but safeguards are needed to prevent misuse or attack.