Mobile App Development

- Binary executable files on the device.
- Can access all API’s made available by OS vendor.
- SDK’s are platform-specific.
- Each mobile OS comes with its own unique tools and GUI toolkit.

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Reusability: Low</td>
<td>High-level services important to personal mobile experience.</td>
</tr>
<tr>
<td>Development &amp; maintenance: Time-consuming &amp; expensive.</td>
<td>Full use of all functionalities that modern mobile devices have to offer.</td>
</tr>
<tr>
<td>Designers are required to be familiar with different UI components of each OS.</td>
<td>High usability.</td>
</tr>
<tr>
<td>Upgrade flexibility: Low.</td>
<td></td>
</tr>
</tbody>
</table>
Virtual Machine Approach

- A virtual machine is used to abstract the target platform details from the application’s running code.
- The framework provides both the API and runtime environment.
- The runtime executes on the mobile device and enables interoperability between the device’s OS and the mobile application.

**ADVANTAGES:**
- Improved performance and user experience.
- Full access to functionalities of underlying mobile OS and device specific capabilities.
- Portability: VM’s are easier to maintain & more flexible to extend.

**DISADVANTAGES:**
- Slower due to runtime interpretation latency.

Cross-Platform Development

- Separates build environment from target environment.
- Platform-independent API using a mainstream programming language like JavaScript, Ruby or Java.
- The cross-compiler transforms the code into platform-specific native apps.
- The software artifact generated can be deployed and executed natively on the device.

**ADVANTAGES:**
- Improved performance and user experience.
- Full access to functionalities of underlying mobile OS and device specific capabilities.

**DISADVANTAGES:**
- Highly complex as cross-compilers are difficult to program.
- Need to be kept consistent with fragmented mobile platforms and operating systems available.

Cross-Platform Development

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Reusability</td>
<td>Might not support every feature of OS</td>
</tr>
<tr>
<td>Plugins</td>
<td>Cannot use own tools/IDE</td>
</tr>
<tr>
<td>Easy for web developers</td>
<td>Slower</td>
</tr>
<tr>
<td>Reduced development costs</td>
<td>High end graphics &amp; 3D support limited</td>
</tr>
<tr>
<td>Support for enterprise &amp; cloud services</td>
<td>Vendor lock-in</td>
</tr>
<tr>
<td>Easy Deployment</td>
<td></td>
</tr>
</tbody>
</table>
Mobile Web Apps

- Use standard web technologies such as HTML 5, CSS 3 & JavaScript.
- Features of HTML 5 - Advanced UI components, access to rich media types, geo-location services & offline availability.
- Increasing popularity of HTML 5 in rendering engines such as WebKit.
- Runs on a standalone mobile web browser.
- Installed shortcut, launched like a native app.
- UI logic resides locally; makes the app responsive and accessible offline.

ADVANTAGES:
- Multiplatform support.
- Low development cost.
- Leverage existing knowledge.

DISADVANTAGES:
- Limited access to OS APIs.

UI Design Considerations

Cross-Platform Development Tools
Mobile website

Adaptive design, responsive design, media queries

Making a Decision

- App functionality
- Time and budget
- Target audience
- IT Resources
App Distribution

Comparison

Right Now: Mostly Native Apps
Only 20% of our time on mobile is spent in our browser

HTML5 is meant to work seamlessly across mobile platforms and browsers

HTML5 Web apps can be installed from the Web as icons on your home screen across any phone

Personal diary app: Penzu
Many serious apps are being built around HTML5

Example: tradeMONSTER
Ranked best mobile stock-trading app of 2012 by Barron's

Many serious apps are being built around HTML5:

- Works across all operating systems.
- Many developers familiar with HTML.
- Lower development costs (don’t have to start from scratch for each OS).

HTML5 faces its own fragmentation problem: different mobile browsers

<table>
<thead>
<tr>
<th>Mobile Browser</th>
<th>Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safari</td>
<td>62%</td>
</tr>
<tr>
<td>Android</td>
<td>22%</td>
</tr>
<tr>
<td>Opera</td>
<td>8%</td>
</tr>
<tr>
<td>Chrome</td>
<td>2%</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>2%</td>
</tr>
<tr>
<td>Other</td>
<td>4%</td>
</tr>
</tbody>
</table>
The mobile browsers are inconsistent in their support of HTML5 features

![Bar chart showing support levels for different mobile browsers]

HTML5 apps also don’t work as well offline

<table>
<thead>
<tr>
<th></th>
<th>HTML5</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Mobile browser access to HTML</td>
<td>Native app code and data</td>
</tr>
<tr>
<td>Coding Language</td>
<td>HTML5</td>
<td>Java, Objective-C, etc.</td>
</tr>
<tr>
<td>Pros</td>
<td>- Full code reuse - Single app experience</td>
<td>- Native UI - Full platform capabilities - Substantial code reuse</td>
</tr>
<tr>
<td>Cons</td>
<td>- Limited offline functionality - Inferior UI</td>
<td>Limited to no code use</td>
</tr>
</tbody>
</table>

And Mark Zuckerberg’s criticism didn’t help HTML5’s cause...

“I think the biggest mistake that we made, as a company, is betting too much on HTML5 as opposed to native… because it just wasn’t there…”
—Mark Zuckerberg, September 2012
LinkedIn recently dumped HTML5 for native too...

Comparison Mobile Web & Native

More Pros & Cons

- No centralized distribution (pro & con)
- No money to Google & Apple
- Security “better” for native apps
- Immediate updates & distribution
<table>
<thead>
<tr>
<th>Item</th>
<th>Winner</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rich user experience and performance</td>
<td>Native Apps</td>
<td>HTML5 still faces challenges in ensuring device-native features across all mobile browsers.</td>
</tr>
<tr>
<td>Cross-platform deployment costs</td>
<td>HTML5</td>
<td>The W3C is pushing the web as a publishing, and TV solution to bring the Web to all devices.</td>
</tr>
<tr>
<td>Immediate updates and distribution control</td>
<td>HTML5</td>
<td>Native Apps can update smartphone apps on the go, whereas only one vendor can update HTML5 apps.</td>
</tr>
<tr>
<td>Minimization</td>
<td>Native Apps</td>
<td>Native Apps can update smartphone apps on the go, whereas only one vendor can update HTML5 apps.</td>
</tr>
<tr>
<td>Available programming expertise</td>
<td>HTML5</td>
<td>Native Apps currently offer greater ease of access to advanced security features and encryption.</td>
</tr>
<tr>
<td>Fragmentation challenges</td>
<td>HTML5</td>
<td>Native Apps currently offer greater ease of access to advanced security features and encryption.</td>
</tr>
<tr>
<td>Security</td>
<td>Native Apps</td>
<td>Native Apps currently offer greater ease of access to advanced security features and encryption.</td>
</tr>
</tbody>
</table>

**Goal**

- Create a very simple application
- Run it on a real device
- Run it on the emulator
- Examine its structure

**Google Tutorial**

- We will follow the tutorial at: [http://developer.android.com/resources/tutorials/hello-world.html](http://developer.android.com/resources/tutorials/hello-world.html)
- Start Eclipse (Start -> All Programs -> Eclipse)
- Create an Android Virtual Device (AVD)
- Create a New Android Project
Android SDK

• Once installed open the SDK Manager
• Install the desired packages
• Create an Android Virtual Device (AVD)
Creating a Project (1)

Creating a Project (2)

Need the items circled

Then click Finish

Package Content

All source code here

Java code for our activity

Generated Java code Helps link resources to Java code

Layout of the activity Strings used in the program

All non-code resources

Images

Android Manifest
Android Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.helloandroid"
android:versionCode="1"
android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".HelloAndroid" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Activity

- An Android activity is focused on a single thing a user can do.
- Most applications have multiple activities

Activities start each other
Revised HelloAndroid.java

```java
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
tv.setText("Hello, Android – by hand");
        setContentView(tv);
    }
}
```
/res/values/strings.xml

<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello World, HelloAndroid - by resources!</string>
<string name="app_name">Hello, Android</string>
</resources>

HelloAndroid.java

package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {

    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}

Introduce a bug

package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Object o = null;
        o.toString();
        setContentView(R.layout.main);
    }
}
Run it!

References

• This tutorial is a brief overview of some major concepts... Android is much richer and more complex
• Developer’s Guide
• API Reference

XML

• Used to define some of the resources
  – Layouts (UI)
  – Strings
• Manifest file
• Shouldn’t usually have to edit it directly, Eclipse can do that for you
• Preferred way of creating UIs
  – Separates the description of the layout from any actual code that controls it
  – Can easily take a UI from one platform to another
**R Class**
- Auto-generated: you shouldn’t edit it
- Contains IDs of the project resources
- Enforces good software engineering
- Use `findViewById` and `Resources` object to get access to the resources
  - Ex. Button `b = (Button)findViewById(R.id.button1)`
  - Ex. `getResources().getString(R.string.hello)`;

**Layouts (1)**
- Eclipse has a great UI creator
  - Generates the XML for you
- Composed of `View` objects
- Can be specified for portrait and landscape mode
  - Use same file name, so can make completely different UIs for the orientations without modifying any code

**Layouts (2)**
Layouts (3)

• Click ‘Create’ to make layout modifications
• When in portrait mode can select ‘Portrait’ to make a res sub folder for portrait layouts
  – Likewise for Landscape layouts while in landscape mode
  – Will create folders titled ‘layout-port’ and ‘layout-land’
• Note: these ‘port’ and ‘land’ folders are examples of ‘alternate layouts’, see here for more info
  – http://developer.android.com/guide/topics/resources/providing-resources.html
• Avoid errors by making sure components have the same id in both orientations, and that you’ve tested each orientation thoroughly

Layouts (4)

Manifest File – Adding an Activity
Android Programming Components

- Activity
  - [http://developer.android.com/guide/topics/fundamentals/activities.html]
- Service
  - [http://developer.android.com/guide/topics/fundamentals/services.html]
- Content Providers
- Broadcast Receivers
- Android in a nutshell:
  - [http://developer.android.com/guide/topics/fundamentals.html]

Activities

- The basis of Android applications
- A single Activity defines a single viewable screen
  - the actions, not the layout
- Can have multiple per application
- Each is a separate entity
- They have a structured life cycle
  - Different events in their life happen either via the user touching buttons or programmatically

Services

- Run in the background
  - Can continue even if Activity that started it dies
  - Should be used if something needs to be done while the user is not interacting with application
    - Otherwise, a thread is probably more applicable
  - Should create a new thread in the service to do work in, since the service runs in the main thread
- Can be bound to an application
  - In which case will terminate when all applications bound to it unbind
  - Allows multiple applications to communicate with it via a common interface
- Needs to be declared in manifest file
- Like Activities, has a structured life cycle
USB Debugging

- Should be enabled on phone to use developer features
- In the main apps screen select Settings -> Applications -> Development -> USB debugging (it needs to be checked)

Android Debug Bridge

- Used for a wide variety of developer tasks
  - Read from the log file
  - Show what android devices are available
  - Install android applications (.apk files)
- In the ‘platform-tools’ directory of the main android sdk directory
  - Recommend putting this directory and the ‘tools’ directory on the system path
- adb.exe

Debugging

- Instead of using traditional System.out.println, use the Log class
  - Imported with android.util.Log
  - Multiple types of output (debug, warning, error, ...)
  - Log.<tag>(<msg>,<string>)
- Can be read using logcat.
  - Print out the whole log, which auto-updates
    * adb logcat
  - Erase log
    * adb logcat -c
- Filter output via tags
  - adb logcat <tag>:<msg type>*
  - can have multiple <tag>:<msg type> filters
  - <msg type> corresponds to debug, warning, error, etc.
  - If use Log.d(), then <msg type> = D
- Reference
Introduction to Objective-C and iPhone/iOS Development

Introduction

• Objective-C is implemented as set of extensions to the C language.
• It’s designed to give C a full capability for object-oriented programming, and to do so in a simple and straightforward way.
• Its additions to C are few and are mostly based on Smalltalk, one of the first object-oriented programming languages.

Why Objective C

• Objective-C incorporates C, you get all the benefits of C when working within Objective-C.
• You can choose when to do something in an object-oriented way (define a new class, for example) and when to stick to procedural programming techniques (define a struct and some functions instead of a class).
• Objective-C is a simple language. Its syntax is small, unambiguous, and easy to learn.
• Objective-C is the most dynamic of the object-oriented languages based on C. Most decisions are made at run time.
Object-Oriented Programming

• The insight of object-oriented programming is to combine state and behavior, data and operations on data, in a high-level unit, an object, and to give it language support.
• An object is a group of related functions and a data structure that serves those functions. The functions are known as the object’s methods, and the fields of its data structure are its instance variables.

The Objective-C Language

• The Objective-C language is fully compatible with ANSI standard C.
• Objective-C can also be used as an extension to C++.
• Although C++ itself is a Object-Oriented Language, there are differences in the dynamic binding from Objective-C.

Objective-C Language (cont.)

• Objective-C source files have a “.m” extension
• “.h” file is the interface file
• For example:
  – main.m
  – List.h (Interface of List class.)
  – List.m (Implementation of List class.)
ID

• “id” is a data type used by Objective-C to define a pointer of an object (a pointer to the object’s data).
• Any type of object, as long as it is an object, we can use the id data type.
• For example, we can define an object by:
  id anObject;
• nil is the reserved word for null object.

Dynamic Typing

• “id” data type has no information about the object.
• Every object carries with it an isa instance variable that identifies the object’s class, that is, what kind of object it is.
• Objects are thus dynamically typed at run time. Whenever it needs to, the run-time system can find the exact class that an object belongs to, just by asking the object.

Messages

• To get an object to do something, you send it a message telling it to apply a method. In Objective-C, message expressions are enclosed in square brackets
  [receiver message]
• The receiver is an object. The message is simply the name of a method and any arguments that are passed to it.
Messages (cont.)

• For example, this message tells the myRect object to perform its display method, which causes the rectangle to display itself
  {myRect display};
  {myRect setOrigin:30.0 :50.0};

• The method setOrigin::, has two colons, one for each of its arguments. The arguments are inserted after the colons, breaking the name apart

Polymorphism

• Each object has defined its own method but for a different class, they can have the same method name which has a totally different meaning.
• The two different objects can respond differently to the same message.
• Together with dynamic binding, it permits you to write code that might apply to any number of different kinds of objects, without having to choose at the time you write the code what kinds of objects they might be.

Inheritance

• Root class is NSObject
• Inheritance is cumulative. A Square object has the methods and instance variables defined for Rectangle, Shape, Graphic, and NSObject, as well as those defined specifically for Square.
Inheritance (cont.)

- Instance Variables: The new object contains not only the instance variables that were defined for its class, but also the instance variables defined for its super class, all the way back to the root class.
- Methods: An object has access not only to the methods that were defined for its class, but also to methods defined for its super class.
- Method Overriding: Implement a new method with the same name as one defined in a class farther up the hierarchy. The new method overrides the original; instances of the new class will perform it rather than the original.

Class Objects

- Compiler creates one class object to contain the information for the name of class and super class.
- To start an object in a class:
  ```
  id myRectx; myRect = [[Rectangle alloc] init];
  ```
- The alloc method returns a new instance and that instance performs an init method to set its initial state.

Defining a Class

- In Objective-C, classes are defined in two parts:
  - An interface that declares the methods and instance variables of the class and names its super class
  - An implementation that actually defines the class (contains the code that implements its methods)
The Interface

- The declaration of a class interface begins with the compiler directive `@interface` and ends with the directive `@end`

```objc
@interface ClassName : ItsSuperclass
{
    instance variable declarations
}
method declarations
@end
```

Declaration

- Instance Variables
  - float width;
  - float height;
  - BOOL filled;
  - NSColor *fillColor;

- Methods:
  - names of methods that can be used by class objects, class methods, are preceded by a plus sign
  - also
  - methods that instances of a class can use, instance methods, are marked with a minus sign:
    - (void) display;

Declaration (cont.)

- Importing the Interface: The interface is usually included with the `#import` directive
  ```objc
  #import "Rectangle.h"
  ```

- To reflect the fact that a class definition builds on the definitions of inherited classes, an interface file begins by importing the interface for its super class

- Referring to Other Classes: If the interface mentions classes not in this hierarchy, it must declare them with the `@class` directive:
  ```objc
  @class Rectangle, Circle;
  ```
The Implementation

#import "ClassName.h"
@implementation ClassName
method definitions
@end
-
makeIdenticalTwin
{
if ( !twin )
{
twin = [[Sibling alloc] init];
twin->gender = gender;
twin->appearance = appearance;
}
return twin;
}

Implementation (cont.)

Example:
@interface Worker : NSObject
|
char *name;
@private
int age;
char *evaluation;
@protected
id job;
float wage;
@public
id boss;
|
Implementation (cont.)

- promoteTo:newPosition
{
id old = job;
job = newPosition;
return old;
}
GCC and Objective-C

- Objective-C is layered on top of the C language
- iPhone & iPad “native” applications are written in Objective C
- The Apple dev kit for Objective-C is called “Cocoa”
- Can be written on any computer that has GCC plus “GNUstep” plug-in
- Apple computers have all of this pre-installed, and also have an iPhone simulator in the XCode IDE

Tools

- Apple: pre-installed with the Cocoa frameworks
  - XCode or GCC in terminal window
- Ubuntu: GnuStep is a free clone of Cocoa
  - sudo apt-get
  - install buildessentials
  - gnu/gnustep
  - gnu/make
  - gnu/stepcommon
  - ib/gnustepสาขาdev
- Windows: http://www.gnustep.org/

iOS programming

- Event driven framework
- Interface Designer has some extra macros for the code that act like hooks to variables;
  - IBAction - trigger events like mouse clicks
  - IBOutlet - captures outputs from events
- These tags are not compiled (don’t affect the code) but sit as an extra before variables that the Interface Designer can see.
Start a New XCode iOS Project

• New Project → iPhone/iPad OS
• Navigation-Based - “navigation controller” as used in Contacts app.
• OpenGL ES - cut-down OpenGL
• Tab Bar – app in style of iPod app.
• Utility – Flipside app in style of stock quote app.
• View-Based – single view that you draw into and display to screen.
• Window-Based – basic app with a main window. (we will use this one for examples.)

Try Out - Hello World

• Watch the Hello World tutorial and build your first iPhone application.
• The tutorial can be found at:

  http://www.youtube.com/watch?v=IvR-t5vEpueQ