Smart Smartphone Development: iOS versus Android

Mark H. Goadrich
Mathematics and Computer Science
Centenary College of Louisiana
Shreveport, LA 71104
mgoadric@centenary.edu

Michael P. Rogers
Computer Science Information Systems
Northwest Missouri State University
Maryville, MO 64468
Michael@nwmissouri.edu

ABSTRACT
In a remarkably short timeframe, developing apps for smartphones has gone from an arcane curiosity to an essential skill set. Employers are scrambling to find developers capable of transforming their ideas into apps. Educators interested in filling that void are likewise trying to keep up, and face difficult decisions in designing a meaningful course. There are a plethora of development platforms, but two stand out because of their popularity and divergent approaches — Apple's iOS, and Google's Android. In this paper, we will compare the two, and address the question: which should faculty teach?

Categories and Subject Descriptors
K.3.2 [**Computer Science Education**]: Computer and Information Science Education – *computer science education*

General Terms
Philosophical, Instructional Technologies, Multimedia, New Curriculum Initiatives

Keywords
iPhone, Android, iOS, Xcode, Eclipse, Objective-C, Java, Smartphones, Mobile Devices, apps

1. INTRODUCTION
Driven by industry needs, student interest, and the relative maturation of the technologies, more faculty are now in a position to consider including modules or even offer full courses in mobile development [10, 12-15, 22, 23, 26]. There are a plethora of platform possibilities — Apple's iOS, Google's Android, Microsoft's Windows Mobile, Nokia's Symbian OS, RIM's BlackBerry, etc. — and while a survey course might be intriguing, the logistics would be daunting, and the failure to develop expertise on any single platform would likely prove unsatisfying. In short, a choice must be made.

A detailed examination of so many platforms is impractical here, so to assist faculty in their most critical first decision, the authors, too, have had to make a choice: we have elected to contrast development on two of the leading platforms among college students [7], Apple's iOS and Google's Android. We will compare their hardware/operating system requirements, Software Development Kits (tools, frameworks, languages, documentation), instructor resources, and finish up by describing the development of a "Hello, World!" app on both platforms.

2. HARDWARE/OS REQUIREMENTS
2.1 iOS
iOS development requires Macintosh computers running Mac OS X 10.6 (Snow Leopard). Since apps tend to be relatively small in size, and run on much slower processors, the computers themselves need not be particularly powerful.

Realistically, hardware may be one of the most irksome impediments to iOS development: while Macintoshes are common on campus, they are less so in Computer Science departments. One solution is to network with colleagues in other departments, or failing that, purchase some low end (but perfectly adequate) Mac Minis, so named for both their dimensions and price.

iOS development can almost be done entirely on the computer, as the simulator that is bundled with the iOS SDK is perfectly acceptable in most situations. The only difficulties arise in usability testing —using the mouse with a simulated touch screen feels unnatural — and apps that require access to particular hardware (GPS, Camera, Accelerometer, Magnetometer) cannot be realistically tested on the simulator. For accurate testing, an actual iPhone, iPad, or iPod Touch (an "iFamily device") is required.

Outfitting an entire class with an iFamily device could be expensive. While an iPhone 3G^s can be purchased for $99, in the United States a minimum 2-year AT&T contract is required, pushing the true cost to over $1000. A more palatable option might be an iPad ($499) or iPod Touch ($199). These lack some of the iPhone's more interesting amenities (built-in GPS, accelerometer, compass, and camera), but those might not be necessary in a beginning class. It would not be necessary to provide a device to each student; either one per team, or just a handful available for checkout would suffice, and perhaps provide an accidental lesson in scheduling algorithms.

2.2 Android
Unlike iOS, which is restricted to Mac OS X, Android can be developed using any of the current major operating systems, Windows (XP or higher), Mac OS X (10.5.8 or higher), and Linux systems (running with kernel 2.6 or higher). This provides great flexibility as almost any modern computer science laboratory should be suitable: no specialized hardware is required.

Developers can create Android Virtual Devices (AVDs), each AVD configured to represent a particular physical mobile device, that run on the Android Emulator [2]. As the name suggests, AVDs allow for extensive testing without the need for an actual phone. An AVD can be instantiated with many different screen resolutions, SD card sizes, and SDK versions to fully test the compatibility of their application over a wide range of Android implementations. GPS readings and phone/SMS interrupt signals can be passed to the emulator through a telnet connection, and accelerometer, orientation and compass readings can be manipulated using OpenIntent's SensorSimulator [21].

As is the case with iOS, testing applications on actual devices is invaluable. Two different development phones can be purchased from Google through an Android Market account, the Nexus One at $529 running SDK 2.2, and Developer Phone 2 at $399 running SDK 1.6. For educational grants, Google also recommends contacting individual hardware manufacturers such as HTC, Motorola and Samsung.

3. SDK

3.1 iOS

iOS apps are written using Xcode, a modern IDE used to code, debug, and lay out the interface. Students familiar with Eclipse, NetBeans or Visual Studio will feel right at home.

For more sophisticated debugging, a separate application called Instruments can be used to detect memory leaks, profile where the app spends most of its time, and ascertain how it utilizes system resources. Both Xcode and Instruments require the hardware described in section 2.1.

Xcode projects can be targeted specifically for iPhones or iPads (the interface is sized appropriately), or written to be universal, running on the iPhone in full-size or on the iPad at a user-selectable resolution.

The iOS SDK is not new. Its origins can be traced to NeXT OS, and while it continues to evolve at a rapid pace to accommodate mobile devices, it has been in widespread use on Mac OS X for over 10 years [11]. This shows in the iOS frameworks, collectively known as Cocoa Touch, which are meticulously designed, exhaustively tested, and thoroughly documented. They provide an excellent example for students to emulate.

Cocoa Touch incorporates numerous unusual, if not unique, design patterns that by themselves are worth studying [5]. For example:

- interface elements, created visually using a graphical editor, are stored in .xib files, then graphically connected to code via *outlets*.
- Object allocation and initialization take place in distinct steps, which reduces the number of initialization methods (equivalent to Java constructors) that must be defined.
- Instead of subclassing, distinctive behavior is achieved by equipping classes with *delegates*. Delegates are simply references to objects that implement a particular protocol, and consulted whenever an object needs to respond.
- blocks (equivalent to lambda expressions and closures in other languages), while not unique to iOS 4, are ubiquitous in the SDK, and provide an even simpler alternative to delegation

iOS applications are most commonly written using Objective-C, a superset of ANSI-C that borrows its OO syntax from SmallTalk. Cocoa Touch provides a rich set of collections, so it is possible to do most coding without having to dwell on the arcana of pointers. However, students must deal with memory management themselves — for performance reasons garbage collection is not available — a bonus for those who spend most of their time in Java or C#.

Perhaps one of the most fundamental design patterns is the separation of interface and implementation, and this is embedded into all iOS apps. In the language itself, an Objective-C class is defined in two files. The interface, in a .h file, contains instance variables, method signatures and properties (simplifying the writing of accessors and mutators). The implementation, in a .m file, contains method bodies. In an Xcode project, the interface is specified in a .xib file, and the implementation in an Objective-C class.

3.2 Android

Android OS is considerably newer than iOS. It has been under rapid development, moving from version 1.0, released in May 2007, to version 2.2, released in May 2010. A majority of Android devices now use 2.1 or above, although a significant portion of devices remain at 1.5 or 1.6 due to hardware limitations [3]. In classroom settings, this fragmentation can either be ignored, with students standardizing on one screen size and OS version, or exploited, to create tangible real-life testing scenarios of cross-platform development.

Eclipse is the recommended and most popular development environment for Android. With the use of the Android SDK plugin, developers can employ this powerful IDE to create projects and skeleton code, push apps to the emulator, and sign apps for release in the Android Market. Although it is possible to develop for Android outside of Eclipse, the benefits of having quick access to the SDK, easy deployment to the emulator or phone, and an integrated debugging environment make it an obvious choice.

The GUI layout for an Android app consists of XML files which include Layout and View elements. Views can be placed hierarchically inside Layouts to create the GUI functionality in either a relative or absolute mode. Eclipse includes a simple WYSIWYG editor for creating views and editing their properties, as well as exposing the underlying XML for manual alterations to the code. Another alternative for layout creation is DroidDraw [6], a platform independent application that can export XML for use in Android projects.

The controller and model functionality of an Android app are written in Java, using a subset of the Java 6 SE API, where the swing, awt, and applet classes have been replaced with custom libraries for graphics and mobile development. Instead of running on a Java Virtual Machine, Google developed its own virtual machine environment for mobile devices called Dalvik. Projects are compiled to run on a Dalvik VM, with each application running inside its own VM on the device. Unless they have had experience with the Java GUI framework, students familiar with

Java from a CS1 and CS2 sequence will not notice a difference between standard Java and the subset available in Android. They do not require prior GUI programming exposure. Mapping functionality can be added by including a separate SDK and obtaining an API key for Google Maps.

There are two other main avenues for developing for Android. Scripting Language for Android (SL4A) provides quick access to the API through Python, Perl, JRuby, among others. Apps using SL4A can be written directly on the phone and used immediately without the need for compiling and exporting an apk (Android package). And at the time of writing this article, Google is beta testing App Inventor for Android, where apps can be written using a graphical language based on LogoBlocks, the same language that underlies Scratch [] and StartLogo TNG. While these significantly lower the entry barrier for development and would be suitable for introductory CS1 courses, upper-level courses specifically geared to application development and mobile operating systems will want to use the full SDK.

4. INSTRUCTOR RESOURCES

4.1 iOS

There are no textbooks on iOS per se, but the number of books geared to professionals is growing, and some could be adopted for classroom use [8, 16, 25]. Apple's technical documentation is vast, precise, comprehensive, and free, and much of it could be used to supplement the textbooks (or serve as the main reference for recent changes in the SDK). A plethora of video tutorials are also available, and some universities are putting entire courses online [4, 28].

While the iOS SDK is free to all developers, it only allows testing on the simulator: however universities can join Apple's free iPhone Developer University Program, which among other benefits permits the installation of the iOS SDK on lab machines, and uploading of apps to iFamily devices.

4.2 Android

As with iOS, there are no explicit textbooks for Android development, however many professional introductory texts can easily be used in the classroom [17, 18]. The official tutorials, development guide, and API reference include numerous basic code examples and complete Eclipse projects, and many tricky coding questions are answered in the StackOverflow forums [20, 27]. In addition, a number of courses on Android have been offered with the course materials publicly shared [1, 9, 19].

5. HELLO WORLD EXAMPLE

To compare the two platforms, we narrate the development of a small "Hello, world!" style app called HelloPermute: each time a button is clicked, the characters of the phrase "Hello, world!" will be displayed in a different order.

Readers interested in seeing the creation of HelloPermute in its entirety may wish to view the videos of the process [24].

5.1 iOS

5.1.1 Creating the Project

When creating a new project in Xcode, the user is offered multiple templates. In this case, we choose one of the simplest, a View-based application, targeted for the iPhone. The template provides:

- a single view, stored in a .xib file, on which we graphically lay out the interface elements of the project;
- a view controller, stored as code in a .h (header) and .m (implementation) file, which plays its customary role as an intermediary between the view and model; and
- a .plist (property list) file, in which we define the app's name, and icon, and other items.

The template does not provide a model. It is simple to add one, but in the interests of brevity, and because our model is almost trivial, we will merely incorporate it into the view controller.

5.1.2 Creating the Interface

Double-clicking the .xib file reveals a window consisting of 3 components — a View, which will display our interface elements; the File's Owner, which represents the view controller; and the First Responder, which references the UI element that first receives an event.

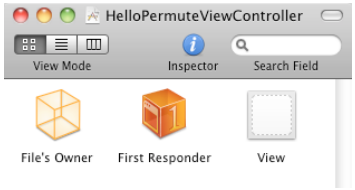

Figure 1: Components of a .xib file

To create our GUI, we drag a UITextField and UIButton from a library console onto the View, center them, and configure them (selecting the font, background, text, etc.).

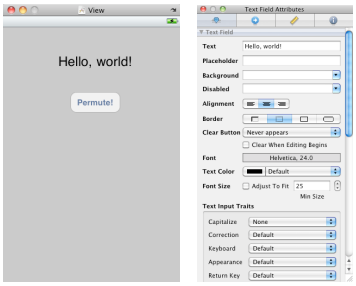

Figure 2: Creating and configuring a GUI

5.1.3 Writing the View Controller

In the view controller class, we must define a method for the UIButton to execute (its *action*), and an instance variable (an *outlet*) so that we can reference the UITextField. Figure 3 shows the interface, HelloPermuteVC.h:

```
// HelloPermuteVC interface, defined in HelloPermuteVC.h
#import <UIKit/UIKit.h>

@interface HelloPermuteVC : UIViewController
{
    IBOutlet UITextField * phraseTF;
}

-(IBAction)permuteWord:(id)sender;

@end
```

Figure 3: The interface of HelloPermuteVC

The syntax may appear daunting at first blush, but it takes just a few minutes in the classroom to explain to students familiar with OO-ideation. Briefly, the class is called HelloPermuteVC, and it extends the Cocoa Touch class UIViewController. The entire interface is enclosed in an @interface … @end block (@ prefaces Objective-C keywords). The interface is divided into two parts: instance variables are defined in curly brackets, and method signatures follow. The instance variable, phraseTF, is a pointer to a UITextField. It is adorned with IBOutlet (an empty macro), which makes phraseTF visible when connecting it to the UITextField object placed earlier.

The method permuteWord, is an instance method, as indicated by the leading - sign. It returns void (IBAction is a macro defined as void). It has one parameter, called sender, of type id (a generic type that can store any reference variable).

Figure 4 shows the body of permuteWord, defined in the corresponding .m file, is as follows:

```
-(IBAction)permuteWord:(id)sender {

    int strLength = [phraseTF.text length];
    NSMutableString * permutedPhrase = [NSMutableString stringWithCapacity:strLength];
    [permutedPhrase setString:phraseTF.text]; // put phraseTF's text -> permutedPhrase

    for(int i=0; i < [permutedPhrase length]; i++)
        [permutedPhrase swapCharsAt:random() % strLength and:random() % strLength];

    phraseTF.text = [permutedPhrase description]; // put permutedPhrase -> phraseTF
}
```

Figure 4: The body of permuteWord, in HelloPermuteVC.m

permuteWord demonstrates the use of *categories*, a powerful feature of Objective-C that makes it possible to add functionality to classes without resorting to subclassing. NSMutableString, as supplied by Apple, does not include swapChars:at:and:. The author added it as a category method, and can consequently be invoked like any other "native" NSMutableString method. The actual code is trivial and therefore not shown here.

5.1.4 Completing the Interface

With the controller class finished, the last steps are to bridge the gap between the code and the user interface, specifically, to associate the method permuteWord: with the UIButton, and the variable phraseTF with the UITextField on the view. Both are done graphically. Figure 5 (left) shows the first step: dragging from the UIButton to the File's Owner causes a small pop-up menu to appear, with permuteWord visible. Choosing permuteWord completes the connection. [permuteWord appears because its return type was an IBAction: had we merely labeled it as void, we would not see it].

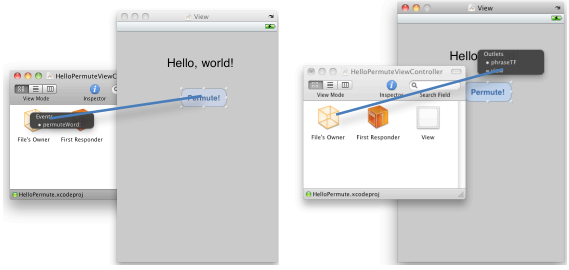

Figure 5: Connecting a UIButton to its action (left), permuteWord, and an IBOutlet to its UITextField (right)

The second step, depicted in Figure 5 (right), is almost the same, except dragging occurs in the opposite direction, from the File's Owner to the UITextField.

It is difficult to convey just how quickly and intuitively one can wire together an application using the techniques described above [24], and most developers embrace it. However, there are times when it is necessary to create a user interface dynamically, and some developers feel more comfortable when they are "in control", so to speak. For those situations and persons, it is possible to eschew .xib files and develop the entire interface in code.

5.2 Android

An Android project has three main elements: the Manifest file detailing its internal organization, resources, such as images and files, and the main Activity and associated class files. The directory hierarchy and a basic template for all three elements are created when a new project is begun in Eclipse.

5.2.1 Project Manifest

The AndroidManifest.xml file, akin to the iOS .plist, stores an overall summary of the app, including the name, version, and minimum SDK necessary to run the app. This is created upon initialization of a new Android project within Eclipse. Resources in other files are accessed with @ notation, such as @drawable/icon. Within this file a developer would also place tags for requesting permissions such as Internet Access, GPS location or Vibrate functionality, and a link to any associated background services. This file can be altered graphically within Eclipse as well as through changing the underlying XML.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
   xmlns:android=
     "http://schemas.android.com/apk/res/android"
   package="edu.sample.hellopermute"
   android:versionCode="1"
   android:versionName="1.0">
   <application android:icon="@drawable/icon"
                android:label="@string/app_name">
      <activity android:name=".HelloPermute"
                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>
   <uses-sdk android:minSdkVersion="1" />
</manifest>
```

5.2.2 XML Resources

Standard resources for an Android project include drawables such as any necessary images and the application icon (png format preferred for transparency), GUI layout, and a strings file commonly used for allowing language independence. Shown first is the /res/values/strings.xml, which names the *app_name*, *hello* and *button* strings.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">HelloWorld!</string>
  <string name="app_name">HelloPermute</string>
  <string name="button">Permute</string>
</resources>
```

The GUI layout is stored in in /res/layout/main.xml. The outer container for the whole application is a RelativeLayout, and placed inside are a centered TextView for displaying the current jumble of "HelloWorld!", centered horizontally, and a Button relatively below the TextView labeled "Permute".

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   android:id="@+id/RelativeLayout01"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   xmlns:android=
     "http://schemas.android.com/apk/res/android">
   <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:padding="10sp"
      android:layout_centerHorizontal="true"
      android:text="@string/hello"></TextView>
   <Button
      android:layout_below="@+id/TextView01"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:id="@+id/Button01"
      android:layout_centerHorizontal="true"
      android:text="@string/button"></Button>
</RelativeLayout>
```

5.2.3 Activity

The HelloPermute Activity file is saved in the source directory as src/edu/sample/hellopermute/HelloPermute.java. The onCreate method takes the place of the usual constructor for an Activity, first initializing the Activity with a call to the super constructor, then inflating the layout from res/layout/main.xml, and finally binding the data member *phrase* to the TextView and *b* to the Button. Unlike iOS's graphical wiring, connections between the GUI view and functionality are linked programmatically. In more complicated applications, an anonymous class is used for the Button's OnClickListener callback detailing the method to be executed. However, since there is only one Button in HelloPermute, we simply implement the onClick method from OnClickListener as part of the Activity, where it grabs the text, shuffles the characters and returns the jumbled String to the TextView. A demonstration of the application running on the Android emulator is shown in Figure 6.

```
package edu.sample.hellopermute;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class HelloPermute
   extends Activity implements OnClickListener {

   private TextView phrase;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      setContentView(R.layout.main);
      phrase = (TextView)
         findViewById(R.id.TextView01);
      Button b = (Button)
         findViewById(R.id.Button01);
      b.setOnClickListener(this);
   }

   @Override
   public void onClick(View arg0) {
      StringBuffer sb = new
          StringBuffer(phrase.getText());
      for (int i=0; i < sb.length()-1; i++) {
         int where = (int)(Math.random() *
                     (sb.length() - i));
         char temp = sb.charAt(i + where);
         sb.setCharAt(i + where,
                      sb.charAt(i));
         sb.setCharAt(i, temp);
      }
      phrase.setText(sb);
   }
}
```

Figure 6: Hello Permute screenshot in Android emulator

6. CONCLUSIONS

Both iOS and Android have their advantages. iOS development requires a specific type of hardware that may be more difficult to obtain — but that might encourage cross-departmental collaboration and expose students to another operating system. The bar for Android is somewhat lower, as development can take place in any modestly equipped computer science laboratory.

Students are much more likely to have seen the combination of Java and Eclipse than Objective-C and Xcode. Again, there is an advantage to exposing students to a relatively novel language and development environment, and a price to pay. Having to confront memory management issues is not necessarily a bad thing, nor is having to work with blocks. But there can be no denying that students studying Android will likely know Java; students studying iOS will very likely *not* know Objective-C. There is, therefore, a modest upfront cost to choosing iOS.

There is easily enough material to devote an entire course to a single environment. However, if curricular constraints make such a course impractical, either could serve well as an intriguing modular programming assignment in an upper-level course. Both are capable of 2D and 3D graphics with OpenGL and database management with SQLite. Students could see practical examples of embedded operating systems and learn to cope with threading, synchronization and locking.

Computing on mobile devices is mushrooming, and regardless of platform choice, a course on the subject will likely be oversubscribed. Either iOS or Android will enable faculty to present the key ideas of mobile computing; strengthen student programming skills that will serve them in good stead regardless

of what platform they end up writing on; and make for an exciting classroom experience.

Table 1. Comparison of iOS and Android across multiple dimensions

	iOS	Android
Minumum Development Operating System Requirements	Mac OS X 10.6	Windows XP Linux Mac OS X 10.5.8
Development Device	$99 iPhone 3G $199 iPod Touch $199 iPhone 4 $499 iPad	$399 Dev Phone 2 (v1.6) $529 Nexus One (v2.2)
IDE	Xcode	Eclipse 3.5
GUI Creation	Xcode	XML
Language	Objective-C	Java (Dalvik) Scripting (SL4A) LogoBlocks (AppInventor)
Reference Website	http://developer.apple.com/iphone	http://developer.android.com/

7. ACKNOWLEDGMENTS

Our thanks to various colleagues at our respective institutions for their support and careful reading of this paper.

8. REFERENCES

[1] Abelson, Hal. *Building Mobile Applications with Android*. 2008. Retrieved from http://people.csail.mit.edu/hal/mobile-apps-spring-08/

[2] Android Developers. Retrieved from http://developer.android.com/guide/developing/tools/emulator.html

[3] Android Developers. Retrieved from http://developer.android.com/resources/dashboard/platform-versions.html

[4] Benson, Edward. *Introduction to iPhone Application Development.* 2010. Retrieved from http://courses.csail.mit.edu/iphonedev/

[5] Buck, E. 2010. *Cocoa Design Patterns*. Addison-Wesley.

[6] Burns, Brendan D. droiddraw. Retrieved from http://www.droiddraw.org/

[7] Digital Media Test Kitchen. 2010. *Smartphone Survey Questions & Results*. Retrieved from http://testkitchen.colorado.edu/projects/reports/smartphone/smartphone-appendix1/#q1b

[8] Dudney, B. 2010. *iPhone SDK Development*. Pragmatic Programmers.

[9] Google Code University. 2010. Retrieved from http://code.google.com/edu/android/index.html

[10] Grissom, S. 2008. iPhone Application Development Across the Curriculum, *The Journal of Computing Sciences in Colleges*, 24, 1 (Oct. 2008) 40-46.

[11] History of Mac OS X. In Wikipedia. Retrieved from http://en.wikipedia.org/wiki/History_of_Mac_OS_X

[12] Kurkovsky, S. 2009. Engaging students through mobile game development. *SIGCSE Bulletin* 41, 1 (Mar. 2009), 44-48. DOI= http://doi.acm.org/10.1145/1539024.1508881

[13] Mahmoud, Q. H. and Dyer, A. 2007. Integrating BlackBerry wireless devices into computer programming and literacy courses. In *Proceedings of the 45th Annual Southeast Regional Conference* (Winston-Salem, North Carolina, March 23 - 24, 2007). ACM-SE 45. ACM, New York, NY, 495-500. DOI= http://doi.acm.org/10.1145/1233341.1233430

[14] Mahmoud, Q. H. and Dyer, A. 2008. Mobile Devices in an Introductory Programming Course. *Computer* 41, 6 (Jun. 2008), 108-107. DOI=http://dx.doi.org/10.1109/MC.2008.200

[15] Mahmoud, Q. H., Ngo, T., Niazi, R., Popowicz, P., Sydoryshyn, R., Wilks, M., and Dietz, D. 2009. An academic kit for integrating mobile devices into the CS curriculum. In Proceedings of the 14th Annual ACM ITiCSE (Paris, France, July 06 - 09, 2009). ACM, New York, NY, 40-44. DOI= http://doi.acm.org/10.1145/1562877.1562896

[16] Mark, D. 2010. *Beginning iPhone 3 Development*. Apress.

[17] Meier, R. 2010. *Professional Android 2 Application Development.* Wrox Press.

[18] Murphy, M. 2010. *Beginning Android 2*. Apress.

[19] Nieh, Jason. *Mobile Computing with iPhone and Android.* Retrieved from http://www.cs.columbia.edu/~nieh/teaching/e6998/

[20] Nurik, Roman. 2010. *Hello, Stack Overflow!* Retrieved from http://android-developers.blogspot.com/2009/12/hello-stack-overflow.html

[21] Openintents. Retrieved from http://code.google.com/p/openintents/wiki/SensorSimulator

[22] Rogers, M. P. 2009. It's for you!: an iPhone development primer for the busy college professor. *The Journal of Computing Sciences in Colleges*, 25, 1 (Oct. 2009), 94-101.

[23] Rogers, M. P. 2010. Wrong number: avoiding the hidden perils in iPhone development. *The Journal of Computing Sciences in Colleges*, 25, 5 (May. 2010), 300-305.

[24] Rogers, M.P. 2010. iPhone App Demo [Video file]. Retrieved from http://www.youtube.com

[25] Sadun, E. 2010. *iPhone Development Cookbook*. Addison-Wesley.

[26] Spertus, E., Chang, M. L., Gestwicki, P., and Wolber, D. 2010. Novel approaches to CS 0 with app inventor for android. In *Proceedings of the 41st ACM SIGCSE* (Milwaukee, Wisconsin, USA, March 10 - 13, 2010). ACM, New York, NY, 325-326. DOI= http://doi.acm.org/10.1145/1734263.1734373

[27] Stackoverflow. Tagged Questions. 2010. Retrieved from http://stackoverflow.com/questions/tagged/android

[28] Stanford University. 2010. CS 193P iPhone Application Development. Retrieved from http://www.stanford.edu/class/cs193p/cgi-bin/drupal/downloads-2010-winter