Mobile Applications in Android

Nick Honetschlager
Computer Science Department
Saint John's University
Collegeville, MN
nhonetschlager@gmail.com

***Abstract* - In today's world, mobile applications are becoming increasingly important in all aspects of our lives. No longer are phones reserved just for making calls, they now do more than the PC's of a few years ago. The open source Android operating system is a great example of the future of mobile applications. Utilizing existing tools such as Eclipse and free plugins, the barrier to entry is low enough for almost any programmer familiar with Java to begin developing. For my demonstration, I utilized the Android platform with the Java programming language to demonstrate the power and ease of use of these tools when used to create a smart phone application.**

I. INTRODUCTION

Not even a decade ago, mobile phones were used almost exclusively for phone calls and the occasional text message. This did not change much until somewhat recently. The number of mobile applications has multiplied with the proliferation of smart phones, and other portable smart devices [10]. The number of common mobile devices has increased also, and this growth has been accelerated even further with better hardware and software. Improvements such as touch screens have also changed development, allowing developers to create more user friendly applications and more interactive games. Android in particular is an excellent platform for those starting out in mobile phone application (app) development. It offers several built in resources such as existing libraries and useful API's. Apps are coded and tested in the familiar Java programming language using the integrated development environment (IDE) Eclipse, and can be debugged on an actual Android device.

II. BACKGROUND

A. Early Beginnings – MTS & Cellular

Mobile telephony had early beginnings in the 1940's, but the era of cellular telephony that we are familiar with did not begin to grow until the 1980's. The first pre-cellular system was known as the Mobile Telephone Service (MTS). It was developed by AT&T, and the hardware was built by Motorola [12]. It had many limitations compared to today's technology. First, the equipment was bulky. It was often paired with vehicles since trunk space could be used for its housing. Second, it drew a lot of power, so batteries had to be very large. This also made it suitable for pairing with a vehicle since it provided a stable mobile energy source. Third, it was difficult to use. The MTS hardware did not operate like a regular telephone. The unit had to be warmed up (prior to transistors), and then the user had to search for a clear channel. This was often difficult due to users competing for few channels on the small piece of bandwidth allocated to the service. Once on a clear channel, users had to contact an operator to be connected to their party. It also had several other disadvantages. Communication was not full duplex so you could not talk and listen simultaneously, other MTS users could listen in on each other's calls, and the service was extremely expensive for its time.

The idea for today's cellular telephony was actually drawn up in the 1940's by Donald H. Ring, a member of AT&T's Bell Laboratories. His idea divided cities into neighborhood-sized coverage areas called cells, and each had its own antenna. To avoid interference, each cellular site used its own set of frequencies. Unfortunately, bandwidth was very limited. However, since there were to be numerous cellular sites, frequencies could be used over again as long as they did border each other [12]. As a user would move across cellular areas, the system would pass the connection from one antenna to the next. This would theoretically reduce power consumption, reduce weak signals, and ease future expansion of cellular coverage [12]. Deployment was held back for several reasons including insufficient computing

power for switching, a lack of frequencies, and bulky hardware. Computing power was minimal in the 1940's, and switching cellular sites takes a lot of computations when many users and cellular sites are involved. When a user switches to another cellular site, the call has to move to another transceiver, and also has to change frequencies. A new channel has to be selected within those frequencies since there are potentially thousands of users accessing a sliver of bandwidth at once. This all has to be done very quickly to maintain the call, and therefore is very difficult to do. That is one of the largest reasons why the concept of cellular phones was shelved for so long. Bandwidth was also very limited for this service. It took AT&T and Motorola decades to lobby for just a few MHz. It took them time to prove to the Federal Communications Commission (FCC) that this concept was worth the space on the spectrum. Hardware also did not shrink enough until the 1970's to produce a prototype. Radios prior to the transistor in the 1940's still used large tubes and were impractical for anything besides a car mounted unit.

B. Cellular Phone Era

Mobile cellular handsets date to the 1970's when they were just a prototype by Motorola. Cellular phone evolution can be broken down into four distinct phases: brick, candy bar, feature phone, and smart phone eras. Pressured to design a working handheld mobile phone before AT&T, Motorola developed a working prototype in only a few months [14]. They had most of the miniaturized components on hand and technology to produce one, but prior to that everyone had been focusing on car phones where size was less of a concern. However, these early models were still bulky and carried large batteries just to have enough power to reach the few cellular towers available. Only a few short calls could be made on one charge, and there were no other features available. The first model marketed by Motorola was the Dynatac 8000x [14]. Due to their large size, weight, and boring design, these early cellular phones are classified into the "brick" era.

The "candy bar" era introduced the boxy phones reminiscent of the 1990's that had a characteristic rectangle shape that gave them their name. These phones were much smaller than the previous generation's due to the increased number of cellular towers available (did not need as strong of transmitters). They had capabilities beyond that of the previous generation, and included such things as Short Message Service (SMS or "texting"). These devices also welcomed in the second generation of phone network technology, or "2G" [9].

The "feature phone" era incorporated more functions into mobile phones such as taking pictures, listening to music, and basic internet access. Unfortunately, several of these "advanced" features were unused by a majority of the population. However, these devices were slimmer and more attractive to consumers than the previous generation. They may not have made quite the technological leap of the previous generation of phones, but they opened the doors for the application filled ones of the succeeding generation. A shift in network technology continued, and this change introduced packet-switching on cellular networks. Experts consider this time as being "2.5G" [9]. This leads into the next era which is the coming together of phone abilities and Personal Digital Assistant (PDA) type functions.

PDA's, were the devices that current smart phones borrow much of their functionality from. Most people credit Apple with the first PDA, who actually coined the term. Their device was known as the "Newton", and performed tasks such as holding calendar appointments, address books, personal notes, etc. Later on in 1996, 3COM came out with the Palm Pilot, and Microsoft with their Windows CE devices. Palm currently makes up a majority of the remaining PDA market with its devices that run Palm OS, and Microsoft with Pocket PC's that run versions of Windows Mobile (currently at version 7). These devices gained popularity during the late 1990's and early 2000's. They are less used by the general population, but are still used in several settings, including the medical field [11].

The "smart phone" era began as far back as the early 2000's. Smart phones are phones with advanced features such as those founds on PDAs and PCs. These early devices were the sometimes poor combination of cellular phones and PDA functions. At first, they had trouble gaining a share of the cellular phone market. Eventually these devices evolved, and combined the more useful features of a PDA with the benefits of calling and SMS. Some of the more prominent manufacturers were Palm, Nokia and RIM (Blackberry). They had the right combination of superior features and design [9]. Many contained much faster processors and memory capabilities beyond those of previous phones. With this generation of devices came the third generation of network technology (3G), and faster data transfer for these internet ready devices. Some consider the next era to be "touch screen". I consider this to be part of the smart phone era because there is so much overlap, and both are helping to bring in the fourth generation of networks (4G). Both smart phone and touch screen devices utilize the diverse number of operating systems and applications available.

C. *Current Smart Phone Platforms*

In today's market, there are several different operating systems available to the mobile developer: Android, iPhone OS, Blackberry, Symbian and Windows Mobile, just to name a few. The iPhone OS is probably one of the most visible in today's market. It supports a variation of Objective-C for its programming, and Apple supplies a free SDK for its development. Blackberry has its own proprietary OS and software that runs efficiently on its devices. It has a less developed development community than the other platforms. Symbian is an open source operating system that was acquired by Nokia in 2008. Its native language is a variation of C++ that is surprisingly not compatible with other implementations of C++. Nokia somewhat recently created a foundation for it to maintain its open source status. Windows Mobile is a mobile operating system developed by Microsoft. It has three versions with one for smart phones, PDAs with phone functions, and PDAs without phone functions. It is usually programmed in C++, or with a compact version of the .NET framework. Its newest release is version 7 released in 2010 [7].

III. TECHNICAL UNDERPINNINGS

A. *Getting Started*

To begin programming in Android, several things are needed. Fortunately, all software is free, and the only cost is hardware. A modern PC is mandatory for development, but the requirements are not too stringent. Any recent multi-core processor paired with a few gigabytes of RAM will suffice [3]. A high-speed internet connection is also needed, because setup will require downloading Eclipse, the Java Development Kit (JDK), Android Software Development Kit (SDK), and the Android Development Tools (ADT) [3]. Installing and configuring these will be most of your setup time. There will be a few other things to setup such as path variables, and the app emulator.

As stated, you will need the JDK, the Android SDK, and the ADT. They are necessary because they provide all of the functionality for development. The Android SDK provides a complete set of tools for the developer, and brings in functionality not normally found in Eclipse. This includes a debugger, Android libraries, a handset emulator, full documentation, code samples, and tutorials for learning [1]. The JDK is needed for any development in Java. It is the basic code and libraries used for development [3]. The ADT eliminates tedious tasks that most app developers would rather not have to worry about. It takes care of such tasks as building the correct file structure for apps and creating necessary base files [3].

Eclipse makes Android development much easier. It brings together all of the benefits of the JDK, Android SDK, and the ADT [3]. A great feature is being able to compile your code as it is written, and see errors as they arise. User interface design is made easier with several drag and drop tools. Automatic code is written in the background as you use Eclipse to design the layout of your app. Also integrated into Eclipse is the handset emulator for simulating applications on a mobile device. This is a good alternative to the usual text output in a console or a popup user interface window. Instead, this emulator mimics a phone's layout and runs your app within the virtual phone's "screen". It also emulates a Linux kernel on your computer to run your app [3]. This is why the tool can take a long time to initialize, but it is worth the wait if you do not have an Android device to test on.

Visual debugging is also a major advantage of using Eclipse. Similar to the tool integrated into other products like Visual Studio, you can step through a running program line by line to see where an error is occurring or see where particular actions take place within your code. This is standard for any version of Eclipse, but the difference here is that you can debug your program while running it on your connected Android device. This allows a developer to see exactly how their software runs on the particular hardware. Testing on Android is straightforward and it uses the popular JUnit test framework. It uses a simplistic managed test environment that automatically builds and destroys every time it runs, so there is a reduced chance of testing problems [4].

Building an Android app may seem somewhat carefree from what I have described so far, but you must be careful to avoid simple mistakes that will waste development time. Developers must be careful to choose the correct target version. This refers to the different versions of Android available on different phones. Currently, the largest market share is phones with either version 2.1 or 2.2. At the moment, the newest version is 3.0, but very few products support it. This is important because if you target a version higher than what a phone can support, that app will not be able to run on that phone even if it only uses older features. It is safer to target earlier releases because newer versions are backward compatible with older ones [3].

B. *Android – Under the Hood*

Android uses familiar programming languages like Java, XML, and C/C++ for its applications. It uses Java for most of the code, but not the same Java that you are familiar with from

coursework. This version excludes libraries that are unnecessary for mobile devices [3]. Despite this difference, programming in Android should be the same. This different version is called the "Dalvik Virtual Machine" [3]. XML code is used primarily for the user interface side of development. This is aided by the drag and drop interface provided within Eclipse. To do more complicated things you will have to be familiar with editing XML. C/C+ are not used much for creating applications, but are for the libraries. The average developer will not need to worry about these, because they are usually accessed through some type of Java interface [6].

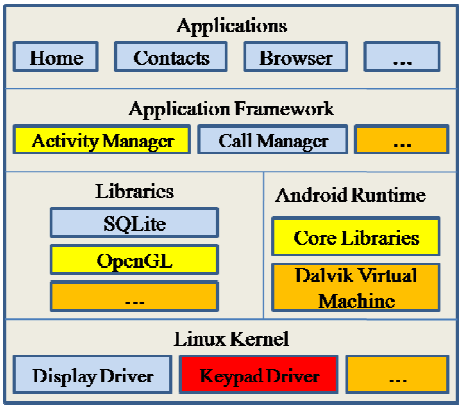

Figure 1 – Android Structure.

1) Structure

The operating system is based off of a modified Linux kernel [2]. This is actually part of a larger architecture that includes several layers of a software stack. The Linux kernel is on the bottom followed by libraries, runtimes, and applications sitting above that. The application layer runs the standard Java applications that most people will interact with. The framework layer provides a basic structure for all applications that run on Android. It can share functions used by multiple applications that are already running, and includes managers for the differing applications. These are all written in Java as well. Below that are the included libraries, and Android has several of these available. However, these are not written in Java like the higher level applications that utilize them. They are written in C and C++, and require a Java interface to access them [8]. These are better known as the Android API's [7]. They include things like graphics, media codecs, and database storage code. The runtime also sits on this layer and is comprised of a custom virtual machine and the core libraries. The core libraries provide most of the code that Java needs to operate. The Dalvik Virtual Machine as mentioned above is a custom version of the usual Java virtual machine. It eliminates the core libraries unnecessary for mobile devices. The virtual machine's function is to translate the Java code to something the operating system can understand [6]. After the code conversions take place the software is run on the device. Each running process has its own virtual machine to avoid bringing down the entire device during a software crash [6]. The final and lowest layer is the Linux kernel and a set of drivers for the hardware components. These include drivers for the display, keypad, and connectivity for WiFi or cellular signals [7].

2) Activities and Services

Android was designed to handle the tasks of managing mobile devices and to do it with very limited resources [5]. Limited resources in the case of mobile devices would be battery power, processing power, memory, and storage. The Dalvik Virtual Machine was meant to run on a slow CPU, with little RAM, and on an OS with no swap space [1]. Android's activity and service life cycle has different methods that make sure that resources are not being wasted on unnecessary or dead activities.

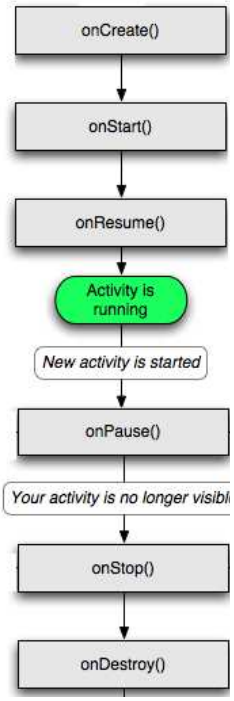

Figure 2 – Android activity and service lifecycle.

An activity starts with onCreate(). This is where needed files are created, and information from the last session is opened [5]. OnStart() is called next and checks to see if the new activity can become the main activity. If it can, control is given to the onResume() method, otherwise control transfers to onStop(). OnResume() retains control as long as the program is running and is in the foreground. If an activity is stopped or pushed to the background, onResume() is called when it is to be continued. In the situation that your activity is pushed to the back, onPause() is called right before it is suspended. This is an important method because it preserves your activity while freeing up system resources for the activity in the foreground [5]. It is important to note that just because your activity is

paused, there is no guarantee that control will ever be transferred back to it. It may even be terminated to make room for a higher priority activity. OnStop() is called after onPause() has been called on your activity and it has moved to the background. From here it may either be resumed later or destroyed. If your activity is done running or is being dumped to free resources, onDestroy() is called to completely eliminate it [5]. Using these methods keeps valuable system resources available to activities that need them, and allows more complex programs to be run on smaller devices.

Services have a similar structure to that of activities, but there are a few differences that become obvious once explained. Methods like onResume(), onPause(), and onStop() are unnecessary because services always run in the background, and require no user interface [5]. There is also a method called onBind() that creates a persistent connection to a service. This is used by an activity to access data being provided by the various services such as GPS information. For a service, onDestroy() is called when no more clients are trying to start a service or bind to it. These methods also help eliminate inactive services that can accumulate system resources [5].

C. Coding an App

1) Built-in Code

There are several different applications and services built into Android that are available to the developer, and can be accessed using relatively simple code. These include email, maps, contacts, calendar, and more [6]. These can all be modified, combined and changed in any way to create custom applications [3]. The ones we are interested in are the location and mapping tools. These are better known as location based services (LBS). LBS use several different methods to determine a mobile device's location. The first, and probably one of the older methods is "Cell ID". Cell phones are constantly "pinging" the cell tower closest to them to maintain communications. Each tower has a unique ID, and they know their precise location. Cell towers have a relatively small radius of only several miles. Since they are constantly pinging these cell towers, they can base their approximate location off of them [5]. Another method of accessing location is triangulation. If a cell phone is in range of two or more towers, by using several of these it can approximate its location [5]. The final method for making LBS possible is GPS. This method by far gives the most precise location and altitude, but there are a few downsides. It increases production costs in devices, uses a significant amount of battery life, and requires a direct view of the sky [5]. This can be a problem if you are indoors or want to preserve your battery. However, the GPS is still one of the most popular methods.

2) Coding a Simple Map

On your first app, it is usually easier to only work with one or a few Java classes if your app is relatively small. This is usually done by creating a class called "MainActivity". This is the main class that is called when your app is started, and can be setup to do this in the project settings. This class has several methods by default such as onCreate(), onResume(), and onPause(). These are the same methods discussed in Android activity lifecycle. Each of these methods requires a default call to the super class in addition to custom code. In our case, most of the code will reside in onCreate(), since this is where most of the program will be created when it is started. OnResume() and onPause() should have code for preserving the state of the app since these are methods that are called when the app is either being pushed to the back or being called back to the front. Since a paused app may sometimes be terminated, it is good to take care of storing important information in onPause(). OnResume() can also contain code to update location information if LBS are being used or triggering the user interface to reflect other changes.

The best way to access the included maps code is to use a MapView or MapActivity. Most people initially think this is Google Maps, but it is a variant of it. The actual Google Maps can be integrated into any application, but you are not allowed to modify it. That is where the usefulness of MapView and MapActivity come in. Google requires you to register with them when you use this, and they give out an API key. The MapView library must also be added as one of the used libraries into the project. Initializing a map view in the code is quite simple to do. The developer must create a new MapView object by referencing it in the main.xml file and assigning it to an object. This is done similar to any other assignment statement in Java. The real wealth of options comes in with the instantiation of the "MyLocationOverlay" object. Method calls on this

```
LocationManager myLocation =
((LocationManager)getSystemService(Context.
LOCATION_SERVICE))
.
.
.
mMyLocationOverlay.getMyLocation()

myLocation.requestLocationUpdates(
myLocation.GPS_PROVIDER, 50000L, 5.0f, new
DispLocListener())
```

Figure 3 – Sample code for initial location and receiving location updates.

object allow you to overlay things on the map, and set many other flags to allow things like satellite and traffic views [5].

Accessing location information is needed for any app that uses a map. Accessing the initial location is a simple process. A "LocationManager" object is required, and a method call getting a system service casts it to that object. Then the location overlay object is used to perform the method call to access the initial location and other related information [5]. Updating location information is therefore also a necessary part of apps using LBS. Your current location changes as you move by receiving updates from the LocationManager object. The requestLocationUpdates() method has several parameters that can be set for the type of location provider, frequency of updates, and the listening object for updates. For example, in Figure 3 my code specifically uses the GPS provider for location updates, requests that these updates occur every 50,000 milliseconds, the minimum traveled distance required is 5 meters for an update to occur, and the name of the listener object to call when there is an update [5].

The MapView and MapController objects then use these things to give us the visual output on the screen. The MapController object can have methods called on it that allow it do things like animate to the current location, and set the default zoom based off an integer value. The MapView object also has many simple methods that can be used on it. Simple methods like setSattelite() or setStreetView() can trigger the satellite imagery or the standard street map views. I used these particular methods in my code for one of the menu buttons. I allow my users to change back and forth between standard map and satellite views based off a button press.

```
mc.animateTo(
mMyLocationOverlay.getMyLocation())
mc.setZoom(16)

mvMap.setClickable(true)
mvMap.setEnabled(true)
mvMap.setSatellite(false)
mvMap.setTraffic(false)
mvMap.setStreetView(false)
```

Figure 4 – Code associated with MapController and MapView

Adding menu options is also a simple task in Android. Android has the ability built in, so very little is required of the developer. The method onCreateOptionsMenu(Menu menu) is called, and is passed a menu object. From there, a method call is made to the super class, and menu items are added by performing "menu.add" and passing the four necessary parameters with it [3]. The two most important of these are second parameter which is the "ID" numbers for the buttons, and the fourth which is the string label for it. The second method, onOptionsItemSelected(MenuItem item), is a method that performs a particular action when a button is pressed [3]. It is passed a menu item and bases which case it will choose off of its particular ID. In the case of Figure 4, the "exit" button calls the method finish() to close the app and release the resources it is holding.

```
public boolean onCreateOptionsMenu(Menu
menu){
boolean supretval =
super.onCreateOptionsMenu(menu);
                menu.add(Menu.NONE, 0,
Menu.NONE, getString(R.string.exit));

public boolean
onOptionsItemSelected(MenuItem item){
                switch(item.getItemId()){
                case 0:
                        finish();
                        return true;
```

Figure 5 – Code for creating an "exit" menu button.

Storing all of this information requires the use of a database. Android provides for this with SQLite. It works with the SQLite libraries by encapsulating all the knowledge of the database within the library, and accessing it with Java interfaces. The best approach is to put all of the required logic for querying, updating and creating databases into Java objects that can be called by the classes that need to manipulate information [5]. The required SQL string can be constructed by passing pieces of information from the Java classes into special methods, and then queried against the database by calling another method.

IV. PROGRAMMING PROJECT

My proposed project set out to utilize some of the provided API's and libraries made available with Android. This includes things like the Google Maps functionality and the LBS for getting location. I planned to overlay other related data over the map relative to the particular location. I also intended to get into SQLite for permanent storage on the Android platform. This would be able to store data once the app has stopped. Finally, I wanted to perform a few algorithms on the location data to make some area derived calculations.

In reality, I was only able to come part way on these goals. I found a natural ending point for this project somewhere between getting the LBS to work with the maps and getting overlay items to work. I realized partway through the project that working

with storage would not be feasible in this project due to time constraints, and instead focused my energy on getting it to where it is now.

V. FUTURE TRENDS

Currently there are many programming languages being used on several different platforms. Technology experts list fragmentation as one of our bigger challenges currently in mobile software development [7]. I see this trend continuing due to the many different technologies currently available, and the emergence of future ones. There will not be a single unifying platform any time soon. What I could see happening is the dominance of a few major platforms such as Apple's iOS, Android, and Windows Mobile. But according to projections, Android is predicted to be the market leader within just 3 years [2].

I also see most phones moving toward smart phone and touch screen technologies. Smart phone sales are increasing faster than feature phone sales as of late 2010, with sale increases 70% higher than that of feature phones [2]. In addition to the apps available to them, they have many more features. This includes things such as email, web browsers, GPS, advanced media capabilities and much more. Electronic components are becoming small enough to fit powerful processing capabilities in small devices. However, battery density will need to increase to keep up with the power consumption of these devices. Currently, battery life only increases about 10% per year [7]. That is one of their largest limitations.

I also believe thousands of new programmers will continue to enter the field of app development. There has been an ever lower barrier of entry into software development because the tools for programming are much easier to access [13]. Many of them are free in the case of Android, and Android even offers free publication to their app market [13]. Most people, including myself, cannot afford expensive software suites and this is a large part of Android's appeal. With these free tools available, younger programmers are more likely to try programming. In addition to that, fewer mobile app developers will have a formal computer science background, and many will be self-taught [2]. I believe this would have been true for me growing up if such tools would have been available. I wanted to learn technical things like app development, but did not have the resources available. I find this to be true for others, and believe this will have a significant impact for aspiring programmers.

There has been an interesting development in this area within the last year. In 2010, Google released the App Inventor for Android. It is a visual programming environment that aims to make app development accessible to anyone through a simple drag and drop interface. It consists of designing a user interface with a visual "component designer" and specifying underlying functionality with a "blocks editor" that represents different activities [2]. It definitely lowers the barrier for entering into the world of "software development", and almost eliminates the need for coding. It is a new type of programming, and I believe this will become more common as software evolves.

VI. CONCLUSION

In what started out as a small market for a rich few, mobile communications has grown to be something that almost everyone uses. Things have changed enormously since the first bulky mobile communication devices. From the early devices that had to be mounted in car trunks, mobile phones have evolved into much smaller and efficient hardware.

In just the last few years the mobile world has changed significantly, and we are now performing tasks on mobile devices that we never could previously. More and more computing power is being fit into smaller devices, and developers are writing complicated apps to take advantage of these functionalities. Android is a solid platform that is helping bring about this change in computing. In its short existence, it has entered into the world of mobile phone platforms and has taken a large share of the market. With its easy to use development tools, low startup costs, and included API's, many more people will enter the world of software development. For me, it was a relatively easy to learn platform for a beginning mobile developer.

In the future, new languages and platforms will likely emerge and technology fragmentation will continue. This will also provide more points of entry for aspiring developers, and more young programmers will likely be joining the community. It is an exciting time to be involved in software development.

REFERENCES

[1] ACM. (2010). Introduction to processing on Android devices. *ACM* , 137.

[2] Butler, M. (2011). Android: Changing the Mobile Landscape. *IEEE Pervasive Computing* , 4-7.

[3] Felker, D. (2011). *Android Application Development for Dummies.* Indianapolis: Wiley Publishing.

[4] Grønli, T.-M., Hansen, J., & Ghinea, G. (2010). Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments. *PETRA '10 Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments* .

[5] Rodgers, R., Lombardo, J., Mednieks, Z., & Meike, B. (2009). *Android Application Development.* Sebastopal: O'Reilly Media, Inc.

[6] Speckmann, B. (2008, April 16). The Android mobile platform.

[7] Tarkoma, S., & Lagerspetz, E. (2010). Arching Over the Mobile Chasm: Platforms and Runtimes. *Computer* .

[8] Xuguang, H. (2009, November 2). An Introduction to Android.

[9] Fling B. 2009. Mobile design and development. 1st ed. Sebastopol, Calif.: O'Reilly.

[10] Oliver E. 2009. A survey of platforms for mobile networks research. SIGMOBILE Mob.Comput.Commun.Rev. 12(4):56-63.

[11] Zurmehly J. 2010. Personal digital assistants (PDAs): Review and evaluation. Teaching with Technology 31(3):179-82.

[12] Farley, T. 2007. The Cell-Phone Revolution.AmericanHeritage.com 22(3)

[13] Bultan, T. (2010). Software for Everyone by Everyone. Santa Barbara, California, USA.

[14] Wolpin, S. (2007). Hold the Phone. AmericanHeritage.com 22(3)

A. REFLECTION

My area of research, application programming in Android, was largely affected by several of my previous courses in the Computer Science Department at Saint John's University. First, I used my programming knowledge that I picked up from the CSCI 161 course. It is an intro level course that introduced me to programming in Java. CSCI 162 continued this experience in programming and taught me further about various types of data structures. CSCI 230 was the biggest benefit for this project. This class helped develop my programming and problem solving skills the most. It was the course that taught me a lot of new things about programming in Java, working with a team and common software development practices. It tied all of these ideas together and many concepts from previous courses. Finally, my time as an intern last summer at Country Financial in Arden Hills, MN helped solidify some of my application development skills from working with an experienced team for several months.

As mentioned above, I benefited significantly from my coursework in CSCI 161 and 162. These two courses gave me the basic foundation on which the rest of my computer science knowledge rests. These two courses taught me programming from the ground up, because I had absolutely no prior experience. I especially struggled through CSCI 161 learning concepts very unfamiliar to me, and trying to understand the object-oriented approach to programming that is used in everything I do today. I also learned to problem solve effectively in this course. The course presented a wide variety of basic programming problems for us to solve, and it adequately prepared me for the larger projects that I would later work on. CSCI 162 continued to help my problem solving skills since we had regular lab meetings. We learned about more complex data structures, and were given the chance to apply them in lab.

CSCI 230 was one of the most beneficial classes to me while at CSB/SJU. This class tied together almost all of my previous coursework and also introduced some new concepts. The biggest benefits from this class was the programming experience I gained from working on the online bank project, balancing between working with a team and alone, and having to teach myself new things as problems came up. There were also numerous other benefits such as learning the Unified Modeling Language (UML) and gaining experience with the Eclipse Integrated Development Environment (IDE). The experience in Eclipse was valuable because it is the environment I used to program my Android application. It taught me useful skills like building a graphical user interface (GUI) with a visual editor, and debugging line by line.

My internship at Country Financial also proved to be invaluable. While I did not program specifically in Java (the language for my project), their use of VB.NET was close enough that I could keep working on my problem solving skills. They also put a lot of emphasis on problem solving for the interns during my time there. We were given the chance to solve a wide variety of programming problems, and were taught the benefits of effectively using a debugger in Visual Studio. This becomes especially important the larger a project gets. Finally, I got to work with senior developers for several months. Sharing their experience and getting to work on real software that they helped me grow as a developer. All of this real life experience gave me much more confidence in these areas.

The current project has tied several of these skills and concepts together, and is improving my understanding of them. In particular, the use of the Java programming language has made the transition extremely easy for this project. Working in Android has also allowed me to further develop my programming and problem solving skills during a semester when most of my other coursework is on paper. I have also gained a better understanding of Eclipse. When I first used it in CSCI 230, I was just being introduced to the software, so my knowledge was fairly limited. From working with Visual Studio at Country Financial, I was able to apply much of that knowledge to my current project.

It is not that my other courses did not contribute to benefiting this project. That would not be true. They have all in some way or another, but the courses I have listed have had the most notable impact, or were the first ones to come along to provide a benefit for a particular skill. Not everything I have learned has been used on this project, but more concepts have been applied than I would have expected. Without the tools I learned from my intro courses to some of the more advanced skills in my upper division classes, every course has played some role in my project, and has been important to my education overall.