Exploring Methods to Develop Cross-Platform Mobile Apps

Rohan Mathur
rohan@rmathur.com
GSET

Vivian Mo
vivian.mo12@gmail.com
GSET

Kavinayan P. Sivakumar
kavinayan.sivakumar@gmail.com
GSET

Jonathan Vielstich
jonathan.vielstich@gmail.com
GSET

New Jersey Governor's School of Engineering & Technology 2013

Abstract

Mobile applications, commonly known as mobile apps, have pervaded modern life and enabled users to solve effectively numerous tasks and problems. Mobile apps are built traditionally in two varieties: native applications such as those for Android or iOS, or web applications that run inside a browser on multiple operating systems. While native applications generally yield higher overall quality when compared to non-native apps and can be built with a much greater feature set, web applications are a popular cross-platform app development method, as they offer quick deployment across platforms. Another type of development platform also exists, for hybrid apps, which incorporates features from web and native applications.

To determine how effective and efficient a hybrid application is at common tasks compared to native and web applications, a test app consisting of six features was written using each platform. The three apps were then tested for efficiency, effectiveness, and limitations. It was discovered that the hybrid application performed as well as the native application in terms of functionality and speed, yet was easier to program.

Introduction

In 1994, the first commercial smartphone was offered for sale. Thirteen years later, Apple released the iPhone, which has since become synonymous with smartphone technology.[1] In the years since the release of the early smartphones, the technology behind them has advanced significantly. Processing speed has increased exponentially and the quality of hardware that can be integrated into the smartphone has increased as well. Growing alongside the popularity of smartphones has been the market for mobile apps. The level of involvement between smartphone manufacturers and their respective software developers has been growing as well. A contributor to this involvement is the incompatibility of the assorted operating systems apps; those written for the iOS operating system cannot be run on an Android operating system and vice versa. As such, developers have to make a decision about which platform to develop for, as learning the native programming language for each operating system is often time-consuming and can be challenging, making this option impractical. Fortunately, it is possible to develop apps that can be run on numerous platforms.

These cross-platform or hybrid apps are deployed to enable developers to reach across these software barriers and expand the availability of their apps. Unfortunately, hybrid apps carry the stigma of lacking the same high-level performance of native apps.[2]

In order to assess the validity of this assumption, an app was developed natively for Android, encompassing many of the most common and most useful functions of the phone. To compare the performance of a hybrid app with that of a native app, Apache Cordova's PhoneGap software, sponsored by Adobe, was used to develop a cross-platform app with the same features as the native app.[3] In doing so, the claims against the abilities of hybrid apps can either be confirmed or disproved. By helping to remove the prejudice against the performance capabilities of cross-platform apps, the general acceptance and use of these apps can increase, allowing for greater involvement and creativity in the world of app development.

Background

When a developer approaches the task of creating an application across platforms, three distinct methods exist that the developer can follow to achieve that. The most common method is to develop these applications with their own native code. In order to develop these native applications, the process is long and arduous. First, the developer must get familiar with the platform's software development kit (SDK), and learn each platform's respective application programming interfaces (APIs). Although this may not seem like much work, there are hundreds of these different APIs that must be learned for each platform. These APIs are also often entirely different on a platform-by-platform basis.

For native Android app development, the main API documentation is available online at developer.android.com, but can be incredibly vague at times, especially for beginners, as the website assumes the programmers have a proficient knowledge of the platform. The situation for iOS and other platforms is quite similar. This online support, along with numerous help forums across the web are where one of the biggest strengths of building a native application comes into play. The number of native app developers willing to help online in addition to the thousands of tutorials on the internet dealing with how to do various tasks in a native app provide both experienced users and beginners alike a way to learn the process of developing an application almost like no other. This key feature of third-party documentation and help sets apart other development methods from native development, solely because of the ease of attaining guidance if the programmer needs it in native app development.

For example, on Android, an API known as a toast notification exists[4] which displays a little message to the user at the bottom of the screen (see appendix for example of a toast notification); on iOS, however, no such API exists. Instead, a developer must use a third-party library to implement functionality similar to this feature in their application. Oftentimes, applications for different platforms are developed in entirely different languages. Android applications, for example, are written primarily in Java, with some eXtensible Markup Language (XML) also used to develop the layouts for the applications. iOS applications on the other hand use a programming language known as

Objective-C,[5] which is almost entirely different compared to Java. Although they share the same general logic structure at many points, the syntax and details of each language is entirely different from one another. Other platforms use different languages as well for their own native apps, adding even more to the confusion, such as Windows Phone 8, which uses the programming language known as C++ for its apps. Although it shares part of its name with Objective-C, C++ has many changes over Objective-C, which classifies it as an entirely different language when compared to Objective-C. This difference in both APIs and programming languages across platforms makes it incredibly difficult and time consuming for a programmer who is looking to develop an application across multiple platforms through a native method.

At the other end of the spectrum, in terms of developing cross platform apps, there exists a type of app known as a web app. These apps are usually simply rich websites (websites that contain more complex content than just plain HTML) written in a wide variety of programming languages. They consist of anything from the traditional HTML 5, CSS 3, and JavaScript combination, to Ruby on Rails - yet another programming language - to web apps dedicated to developing other web apps in their own language, such as a web app by Microsoft Research known as TouchDevelop. These apps can run in the browser of most devices, without compatibility issues, which is helpful for developers who are looking to deploy quickly a single app across multiple devices. This method of developing apps, however, has its drawbacks. Since the web app is run almost entirely within the browser of the device, it does not have access to many integral system functions such as accessing the contacts of the device and more. This severely limits the functionality of the web app. In addition, the web application is often slower than the other choices the developer has as the browser is used to render the app, which runs atop the operating system (OS), instead of just the OS itself.

The final and main format of application that is tested in this study is the type of application known as the hybrid application. This method of developing cross platform applications leverages both the cross platform compatibility of HTML 5, CSS 3, and JavaScript and the feature set of a native application.[6] Each container that the HTML 5, CSS 3, and JavaScript code is dropped into provides functions that offer easy access directly with the device's native application features such as interacting with internal data, while still keeping the app cross platform at its core. This format is the most appealing to developers as it is a mix of the good and bad of each of the two other methods of developing cross platform applications.

Testing and Methods

To compare the different platforms, the efficiencies of a number of tasks must be evaluated in each platform and compared with each other. However, there are millions of apps across all the app stores, and each app has its own feature set. Therefore, testing just one app would neglect many other apps that utilize different features. In addition, finding applications across multiple platforms that have the same functionality as the ones in the other platforms is nearly impossible with the large diversity of applications available in today’s application stores. As a result, the prioritization of the various tasks an application can deliver through its prevalence was the most efficient method to compare platforms. Six tasks were

established as the most commonly used tasks or qualities applications perform:

1. Using an audio player
2. Using the camera
3. Accessing a website
4. Accessing the mobile device's contacts
5. Using a video player
6. Processing speed

Many applications today use a camera, access websites, or play audio or video. In order to include those apps that use a mobile device's internal data, one of the tasks being tested is accessing a mobile device's contacts. Finally, in order to find the processing speed, the code runs a million iterations of two randomly generated integers from zero to ten multiplied with each other. The time that it takes the platform to run these million iterations is a test for the processing speed of the platform.

Android

In order to develop a native Android application, the first step is to download the Android SDK and development environment. Both can be bundled into one single download in a relatively new development environment known as Android Studio (available for Windows, Macintosh, and Linux computers). Android Studio, as seen to the right in Figure 1, is based upon the IntelliJ IDEA development environment. It is an integrated development environment (IDE) geared toward Android Java application development, offering a wide variety of features such as automatic code reformatting to Android development standards, real-time layout previews, autocomplete for many methods and properties, and more. Android Studio is an improvement on its predecessor, known as Eclipse, which in actuality was simply a generic Java IDE with an Android plugin (previously known as the ADT plugin).

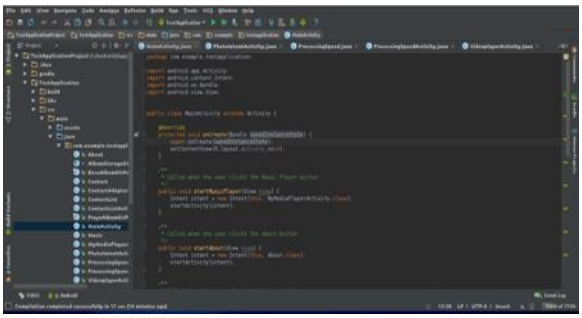

Figure 1: Android Studio's main development screen

Android's main architecture in terms of application loading is focused primarily around the Dalvik Virtual Machine, a simplistic and stripped down version of the Java Virtual Machine (JVM) that runs on most modern computers.[7] Because of this heritage heavily rooted in Java, all native apps are primarily composed of Java for all major programming, alongside XML. The file layout of the traditional Android application source code is very similar to normal Java applications for normal computer applications as well, containing folders such as "res" (standing for resources, where all the images, layouts, and other application resources are stored), and "src" (standing for source, where all the main Java files are stored). Due to the widespread usage of Java in the Computer Science field, the general anatomy of an Android application's source should come as no surprise to experienced developers, even those who are looking at an Android application's source for the first time.

To approach this project's goals of developing a testing application to see how the various functionalities of the application worked, the first step taken was to learn the various APIs that would be needed in the application. The documentation provided on Android's website for developers offers a great overview of the platform and its various features. Due to the construction of

the test app having such a wide number of features, many APIs were needed. The contacts displaying feature, for example, required the learning about the usage of ArrayAdapter (see appendix for example code), which enabled the application to fetch the contacts in the system as an array of values to the application's content displaying method, called a ListView, for usage in the user interface (UI). Another aspect of the API that was utilized in the development of this application was the Android Toast notification system. The system displays a small message to the user giving them bits of information that are important enough to notify the user, but not important enough to have an obtrusive popup. An example of when a toast notification is used is perhaps when the application is refreshing the page in the web browser after the refresh button was hit, after which the application lets the user know that the web browser is reloading the page. These messages required learning the proper way to call the system function, including the parameters of the function such as the message subject, the length of the message (a choice between long or short), and much more. These two APIs, along with hundreds more, are required for the creation of such a test application.

PhoneGap

By utilizing Adobe's online PhoneGap Build service, it is possible to develop a cross-platform mobile app without downloading a new SDK or other software.[8] Adobe offers a free account in which one private app can be made; in addition, unlimited open-source apps can be made by uploading the appropriate files from a public GitHub repository, a website to store open source code using the git version control protocol. While this is sufficient for research purposes, it is possible to build up to twenty-five private apps with either a paid membership or an Adobe Creative Cloud account. The relative ease of this system increases the appeal of using PhoneGap to develop a cross-platform app, as doing so can be accomplished with a program as simple as a default text editor.

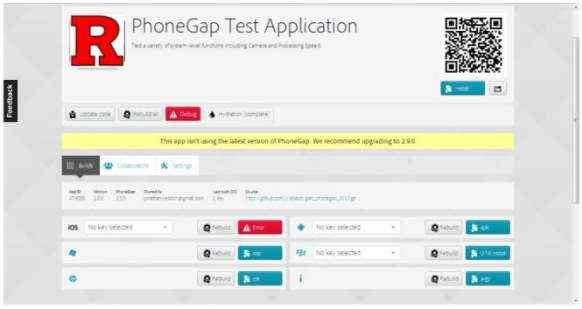

Figure 2: The Adobe PhoneGap Build service

PhoneGap relies on the fact that the main application's content code is written in HTML, CSS, and JavaScript, and that it can be injected into a container for different platforms during build time.[9] As a result, the application can be written just in these three languages and can be used in a variety of platforms such as iOS, Android, Windows, and BlackBerry. The files can be initially created through the command prompt or terminal, after which a blank PhoneGap project is created. The PhoneGap application used for this research was compiled to be used on Android, as they were tested on a mobile device that ran the Android operating system, but these files can be compiled to work with other platforms such as iOS, Windows Phone, and Blackberry by clicking on a different button on the PhoneGap Build service. The files are then stored in a separate repository in GitHub, and then accessed through the PhoneGap Build service.

The first file to be created was the "index.html" file. In the initial development of the app, numerous index files were

created, each containing all of the necessary HTML structure to set up a single process for the eventual final application. A file named "config.xml" was also created, written with XML, to provide all of the metadata for the app, such as version number, as well as the necessary permissions and preferences the app required to function properly. After establishing the base of app, the necessary JavaScript files were created to give the app functionality. Following the successful creation of the various sub-programs for the app, they were integrated into a single app, using HTML to link them together from a new index file. After the functional aspect of the app was completed, the process of fine-tuning the aesthetics began, with numerous modifications being made to the CSS and HTML structures of the app.

The general file structure of a PhoneGap application involves mainly many HTML, CSS, and JavaScript files. The HTML and CSS files generate the general appearance of the application. JavaScript in the source code executes all the tasks, and is essentially the backbone of the content displayed to the user. These files reference a JavaScript file in the PhoneGap files known as "cordova.js", which contains the main PhoneGap code, interfacing with the phone's hardware for an innumerable amount of functions. At build time, this code is added to a container for the respective target platform and then the application is compiled.

As it is a new process of creating cross-platform applications, PhoneGap does not have as much documentation available as other platforms such as Android or iOS. Although there is a help page on the PhoneGap Build website, and a number of YouTube videos on the subject, it is still difficult for a programmer new to the system to start coding efficiently.[10] This does not mean that PhoneGap is impossible to use; in fact, that is far from the truth. PhoneGap is widely seen as one of the best solutions to create hybrid apps. The fact is, however, that there has not been as much work done with PhoneGap as there have been on native apps.

TouchDevelop

TouchDevelop allows the average user to write his or her own apps without the need to download SDKs or IDEs. The user can run the TouchDevelop IDE on a web browser or download the TouchDevelop Windows Phone App. By signing in with a Microsoft, Facebook, or Google account, users can access the "touch-friendly" user interface of TouchDevelop. Then, the user can create and save web apps, called scripts in the cloud. Designed to minimize the amount of typing done by the user, TouchDevelop tries to transfer the act of programming from computer to smartphone.[11]

When a user opens up a script for editing, TouchDevelop provides the user with all the usable programming functions as large, clickable, buttons. On a smartphone, the user is not able to type out words on a keyboard (unless he or she wants to type out a string). With available APIs such as "media" and "senses" displayed when the user is editing, the user no longer needs to be familiar with a language's API to code an app since all the options are presented to the user. By tracing program flow using arrows, simple syntax errors such as missing a semicolon are avoided. Errors are automatically highlighted in red and error messages are displayed. These errors are very descriptive with the issue at hand, making debugging a simple task.

Figure 3: Main development screen of TouchDevelop

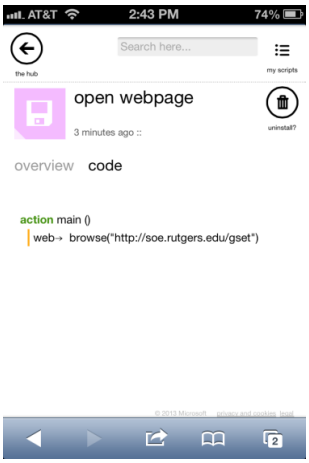

Figure 4: Code for the open webpage script of TouchDevelop

In Figure 3 above, the main development screen of TouchDevelop is shown for the camera app. The action main() contains all the functions of the code. Instead of semicolons, TouchDevelop utilizes arrows to indicate the logic flow. As shown by Figure 3, TouchDevelop provides the usable functions for every new line of code, taking the burden of familiarizing oneself with the functions off the user. There is also a search option available so the user does not have to flip through pages of functions in order to find the one that he or she needs.

The code for accessing a webpage using TouchDevelop is shown in Figure 4. In action main(), the web function is selected. Not only was the web function displayed in the main development screen, all functions also include a brief description. For example, the description of the web function is "web – Search and browse the web." After selecting the web function, the next set of available functions, or functions that work in conjunction with the web function on TouchDevelop, are displayed. From there, the browse function is selected due to its description "browse(url : String) – Opens a web browser to a URL." Not only does this description tell the user of what the web function does, it also prompts to user to enter the URL of the desired webpage as a string.[12]

However, the functionality of TouchDevelop is not limited to its mobile capabilities. By storing scripts in the cloud, users can access scripts using multiple platforms. Since TouchDevelop is developed by Microsoft, its features are optimized for Windows Phone 7, Windows Phone 8, and Windows 8. Still in beta mode, TouchDevelop lacks many features for non-Microsoft platforms such as iOS and

Android. Though the test app requires that the app be able to play audio and video, access to media (songs and pictures stored to the phone) is only supported for Windows Phone at this time. It is also only supported on certain web browsers. For example, TouchDevelop will run on Safari for an iPhone but not Google Chrome.[13] In this experiment, neither a Windows Phone nor Windows 8 was tested. Instead, the features and limitations of TouchDevelop were tested on iOS and Android.

Since it is tailored for the average smartphone user, TouchDevelop has an easy to use API. However, while TouchDevelop is easy to use, it gives the user much less control over the app than if he or she wrote a native or hybrid app from scratch.

TouchDevelop itself is still a work in progress and its use is largely restricted to newer Windows products, so the TouchDevelop community is much smaller than the development community for native and hybrid apps is. That aside, since the API of TouchDevelop is so easy to use, users typically do not need help for app development since the API of TouchDevelop prevents users from pursuing projects that are too complex. As of July 16, 2013, 34,305 scripts have been published, 515 apps have been exported to the Windows Store, and 155 apps have been exported to the Windows Phone Store.[14]

Results and Discussion

Overall, the Android application required the most amount of effort and time from the developer, compared to the other development platforms; over forty-five hundred lines of code went into the application, compared to less than twenty for the TouchDevelop application, or less than one thousand for the PhoneGap application. This is because of a number of factors - mainly the long nature of Java programming, the requirement of importing of libraries, the overriding of certain methods, and more - that are not needed on other development platforms such as PhoneGap (HTML, CSS, and JavaScript) or TouchDevelop. Java is a much more complex language and therefore requires many more keystrokes to achieve the same task that the other applications can accomplish in much fewer lines of code. Java also has many more options that can be tweaked (for example, the web browser has nearly twenty properties that are changed in the code, such as allowing for downloads, setting the loading bar color, and much more). With more effort, however, the code is much more efficient; the code yields much cleaner user interfaces and speed far above that of other applications.

In comparing the development of cross-platform and native apps, native apps possess two clear advantages over hybrid apps: the familiarity of the programmer with the language and the availability of useful documentation. The native app is written in Java, with some minor changes to enable Android functionality, and Java is one of the most commonly used languages. As such, the developers of the Android operating system were safe to assume that a large portion of potential app developers would be familiar with Java. On the contrary, the development of the cross-platform app requires knowledge of HTML, CSS, and JavaScript, which few developers are well versed in all three unless they did any web development in the past, as these are primarily website programming languages. The development of a cross-platform app is likely to require more research, thereby increasing the time it would take to develop the app itself. However, these three languages are relatively simple and easy to learn. In order to develop native apps for

platforms other than Android, a developer would need to be familiar with languages other than Java, the most notable being Objective-C, used for the iOS operating system. This need to learn new languages to create both hybrid and native apps means that hybrid apps remain a viable option.

The remaining advantage of native apps is the extensive documentation available to aid in their development. For Android, there is a website dedicated to the development of Android apps. There are countless other resources available to aid in the development process, found on blogs and commercial sites alike. This simplifies the process of resolving issues with one's app, as other developers have most likely encountered similar issues. The volume of documentation for PhoneGap, however, is significantly smaller. While the PhoneGap website does provide some support, the documentation is relatively disorganized. In order to build properly the camera app, a developer inexperienced with PhoneGap would need to view two separate pages - Camera and Capture - in addition to finding the proper permission to enable the camera function. Despite the high level of interdependency for these functions, one must navigate through the multiple pages of documentation before locating all of the necessary components. Furthermore, the examples provided by the PhoneGap website, while present, are not especially useful, as they do not teach the user how to write his or her code, but merely provide a basic, pre-written program. While some tutorials for PhoneGap from outside resources, such as the video sharing website YouTube, far more tutorials can be found for Android app development, thereby increasing the ease with which one can learn to develop a native application. TouchDevelop has little documentation besides a few "Getting Started" tutorials; these are not necessarily even needed though, as most of the time, the user interface was so easy to use that the tutorials were not even needed.

Logic Structures

In comparing the logic structure of the three developed apps, several key aspects are present in all three. In order for each app to initialize, the correct function must be found within the body of the app's code. In Java, for the native app, this is the onCreate method; in TouchDevelop, action main(); and in PhoneGap, onLoad(). After the app is initialized, the various methods, classes, and functions are called based upon which buttons are pressed. In PhoneGap, this functions as a standard hyperlink, essentially opening a new page in the browser, while in Android, the various methods are called, eventually linking to another portion of the code and activating the appropriate function of the phone. For the camera function, for example, TouchDevelop uses simple commands to activate the necessary functions within the app. The native app calls the user-defined method that will then activate the camera, relying upon the various libraries provided by the Android SDK. The PhoneGap app uses the same type of code as the native app, referencing functions to open the camera applications, but instead of referencing these functions in an SDK, the function definitions can be found in the cordova.js file that makes up the majority of the PhoneGap platform, along with the container that is added at build-time. Logic for the other functions was quite similar as we saw here; it followed the same sort of formatting across the platforms, sometimes almost exactly such as in the case of the processing speed task. This common logic can prove to be very useful when developing an application across platforms as instead of worrying about logic, the programmer can in most cases just switch the syntax to another

language or SDK using the proper APIs of the platform.

Using the Camera

Though camera access is not supported for non-Windows Phones on TouchDevelop, a work-around was engineered using the "choose picture" function under "media." Using "post to wall," the picture is displayed, albeit, distorted on the web app. Despite having found a way to work around the camera function not being supported on Android and iOS, the media → choose picture → post to wall does not save the taken photo onto the internal memory of the phone itself while both the native app and hybrid app are able to save the picture onto the smartphone.

The code for the TouchDevelop applications, unlike the code for the Android and PhoneGap applications, could only be programmed with the same logical processes used in the other applications at certain times due to the restrictiveness of the TouchDevelop API. As a result, the processing speed and accessing the mobile device's contacts were the only code that had the same logical processes as the same tasks in the other platforms.

Limitations of TouchDevelop

Code for the camera and website were already preprogrammed into the TouchDevelop API, and as a result, the code behind each of those buttons was hidden from the programmer. For these two tasks, comparisons done took only into account the functionality of each task, but kept in mind the fact that there is the possibility that each code had a different process. Moreover, since TouchDevelop does not allow for access to media for mobile devices that run operating systems other than Windows and due to the lack of a mobile device with Windows on it, these two specific tasks could not be tested.

Processing Speed

The processing speed task was designed to run a million iterations of two randomly generated integers from zero to ten each multiplied by each other and to find the total time it took the application to perform the task. At first, the hypothesis was that the application would run the fastest on the native platform, as native applications are known for their quality and efficiency. Second would be the hybrid application and last would be the web application. However, when each application was run ten times and the average time was computed, the results were:

Android (Native): 1.4826 seconds
PhoneGap (Hybrid): .3179 seconds
TouchDevelop (Web): 6.6 seconds

Surprisingly, the hybrid application had a faster processing speed for the task than the native application. The answer why lies in the fact that the JavaScript implementation, which PhoneGap is written in, uses fewer data structures and attacks the problem in a more direct manner than Java.[15] However, although the result shows one faster than the other, both programs used the same logic. In this respect, writing a program with many algorithms is virtually the same in both hybrid and native, as only the syntax of both applications differs. In this particular case, this applies to TouchDevelop as well. In general, all of the popular and widely used programming languages use the same flow of logic for algorithms and logical operations.

Internet Access

For the task of accessing the web browser, all three platforms once again had the capability to do so. However, while the PhoneGap and TouchDevelop programs had a relatively small amount of code, the

Android program was substantially longer. Although all three platforms performed the task efficiently and correctly, the PhoneGap and TouchDevelop programs were much easier to write. The reason why this occurs is due to the particular advantage of a hybrid and web app in the fact that they do not need to create a container for the browser.

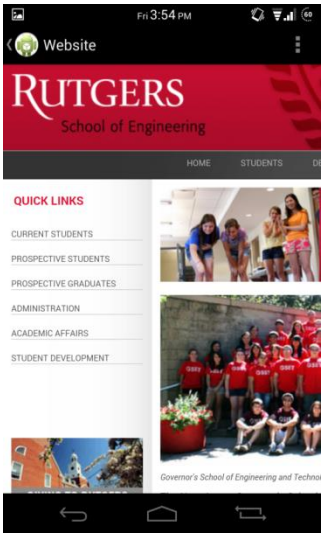

Figure 5: The web display in the native application

Accessing Internal Data

Many applications present in the app market today, regardless of function or purpose, take data from the mobile device they are running on in order to run a task as part of the app. To measure the capabilities of each platform in its regard to this task, code was written in each platform to access a mobile device's internal contacts. Once again, since TouchDevelop is still in beta mode, access to internal contacts was not supported on iOS or Android. Despite this, it is possible to use the "dial phone number" function to call a number already stored as a contact and have the contact name be displayed when the call is placed.

Both the PhoneGap and Android applications were able to access successfully the contacts of the mobile device, as they implemented some part of a native platform.

Using Media Files

In order to take full advantage of the capabilities of modern smartphone technology, an app must be able to play a myriad of audio and video files. Due to the limitations of TouchDevelop on non-Windows platforms at this time, it is impossible to play media files on Android or iOS mobile devices. PhoneGap, however, is quite capable of doing so, provided that the correct permissions are supplied; the same is true for the native app. Therefore, the comparison between the two in this arena must be made based upon speed and ease of use. The PhoneGap app takes a significantly longer time to load the desired files, and may fail to find initially the desired files. The native app, however, is fully integrated with the actual operating system and is therefore faster and more convenient to use. This is a significant advantage for native code, as any app that is created with the intent to use media files must do so swiftly and effectively.

Layout

There is an obvious difference between the layouts of the native and the PhoneGap apps. The native app, which is able to take full advantage of the ability of Android Studio to integrate seamlessly with the Android operating system, is smooth and polished (as seen in Figure 7). The PhoneGap app is only able to create a display based upon the functions of HTML and CSS. As such, the format is limited to what can be easily produced with CSS and HTML. This is evidenced by the format of the header and the buttons on the main pages of the apps. The buttons extend only as far as the words contained within them; the header is simply a black bar (as seen in Figure 6). This can be changed via CSS and HTML attributes but it is harder to do that compared to simply dragging and dropping in a button in Android Studio which

contains an easy-to-use interface to create the application's GUI.

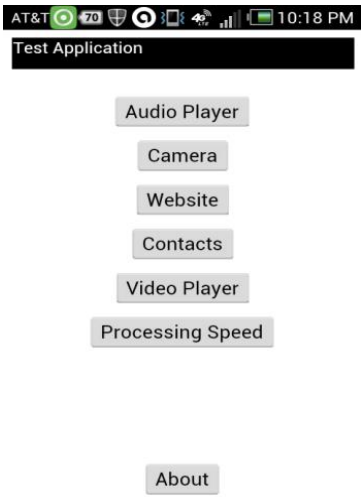

Figure 6: The main page of the PhoneGap app

Some visual HTML and CSS text editors exist, such as Dreamweaver, but they do not integrate as seamlessly into the application's source as Android Studio does. The stylistic discrepancy arose from the nature of the development methods. Within the native app, the code is written with the specific Java functions that enable the Android display to be controlled across various devices; the buttons can be set to fit to any screen size and orientation with a few simple lines of code. Within the cross-platform app, the code is written with general HTML functions that are limited; to achieve the same aesthetics, more complexity would be required.

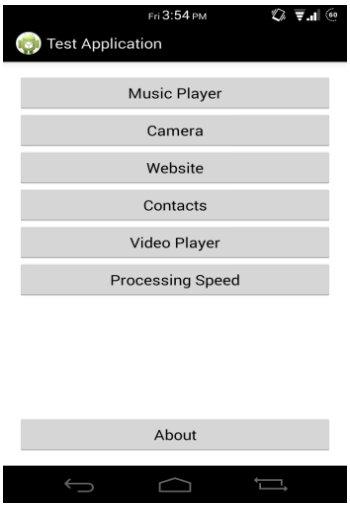

Figure 7: The main page of the native app

Real-World Applications

The data collected from this research can be implemented to a situation in which a platform is being chosen to write a program in. For any programmer wishing to develop an app over multiple platforms, hybrid apps, especially written in PhoneGap, are the most efficient method. Although a programmer can write his or her application over the native platforms across many platforms, it would require a lot of time as well as effort to write the same code in a different syntax and language. However, if writing a program with a hybrid platform, the programmer gets all of the basic tasks a native platform can perform, with the only cost being aesthetics. Even in this case, the programmer has the ability to format the application with the use of CSS alongside the HTML and JavaScript to improve the overall look of the application.

Improvements

Some further work can be done on this research. The processing speed function of the apps can be improved by optimizing the native application for multi-core usage. This would allow for better comparison of processing speed across the different development platforms. Other hybrid application platforms could be tested as

well, instead of just PhoneGap. Although PhoneGap is the most popular, many other cross-platform mobile development software exist, such as Sencha Touch and Intel XDK. Apps are also not the only type of software available for mobile devices; games are becoming more popular as well, which opens up a new aspect of cross platform mobile application development to be researched. Game development is vastly different than normal application development, requiring the usage of APIs such as 3D drawing and more. Finally, a Windows Phone device could be used to test the Windows Phone-exclusive features in TouchDevelop, to fully see whether TouchDevelop could be a viable alternative to hybrid development options (both for testing the media features of TouchDevelop and for exporting the scripts as full applications).

Conclusions

Native apps, hybrid apps, and web apps all have their advantages and shortcomings. Native apps are tailored specifically for a smartphone's operating system, but they take longer to write and the developer must learn a different language for each platform while familiarity with HTML, JavaScript, and CSS are all one needs to develop a hybrid app using PhoneGap. Even though a hybrid app created on PhoneGap can run on all the major platforms - Android, iPhone, Windows Phone, and Blackberry - hybrid apps cannot be tailored to take advantage of a platform in the way that a native app does. Native versus hybrid development is essentially a question of quality versus time but both are viable options for app development from beginners to professionals. PhoneGap, although lacking in documentation compared to native development, is made in such a way that an easy language collection can be used to fill the application with content and therefore can be learned easily. It cuts down the time needed to bring an application across multiple platforms, and therefore, is the best choice for a developer looking to make their application cross-platform. Although it takes a little bit more effort to get the application looking as good as the native application, it matches the native application in functionality and in general is easier to write.

Web apps, such as TouchDevelop, are still in their infancy, and an interesting topic to explore in the future would be the effect of TouchDevelop on the mobile application developing landscape. At the moment, though, in its limited feature state, it is not a viable option for creating a cross platform application. As each of these different development platforms develop further in the future, these options should be assessed as they are all constantly being improved upon and changing at a rapid pace.

Acknowledgements

We acknowledge our mentor, Dr. Vinod Ganapathy for his guidance on this project. We would also like to extend our thanks to our Resident Teaching Assistant mentor, Neha Desai, for her continued support throughout the project, as well as Daeyoung Kim, whose technical expertise was a great resource for this project. We also thank the Director of New Jersey Governor's School of Engineering and Technology, Ilene Rosen, and the Assistant Director, Jean Patrick Antoine. Finally, we would like to thank our sponsors, Rutgers University, the State of New Jersey, Morgan Stanley, NJ Resources, South Jersey Industries, PSE&G, and the GSET alumni community.

We would also like to acknowledge the following people for their contribution through open source software to parts of the applications discussed in this paper:

Sony Arouje – Native Contacts List
http://sonyarouje.com/2011/07/29/list-contactsmy-first-android-app/

Android Developer Team – Native Camera Interfacing
http://developer.android.com/guide/topics/media/camera.html

PhoneGap Team – PhoneGap Audio Player/Camera/Contacts
http://docs.phonegap.com/en/2.0.0/index.html

References

[1] M. Lala, *A Short Look at the History of Apple's iPhone*, WWW Document, (http://www.geekinsider.com/2013/03/05/a-short-look-at-the-history-of-apples-iphone/).

[2] D. Seven, *What is a Hybrid Mobile App?*, WWW Document, (http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-).

[3] *About the Project*, WWW Document, (http://phonegap.com/about/).

[4] *Toasts*, WWW Document, (http://developer.android.com/guide/topics/ui/notifiers/toasts.html).

[5] *About Objective*-C, WWW Document, (https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html).

[6] D. Seven, *What is a Hybrid Mobile App?* WWW Document, (http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-).

[7] *dalvik: Code and documentarian from Android's VM team*, WWW Document, (https://code.google.com/p/dalvik/).

[8] *Get the Android SDK*, WWW Document, (http://developer.android.com/sdk/index.html).

[9] *Phonegap FAQ's*, WWW Document, (http://phonegap.com/about/faq/).

[10] *Topics about the product: PhoneGap Build*, WWW Documents. (http://community.phonegap.com/nitobi/products/nitobi_phonegap_build).

[11] N. Tillmann, M. Moskal, J. de Halleux, M. Fähndrich, *TouchDevelop: Programming Cloud-Connected Mobile Devices via Touchscreen*, WWW Document, (http://research.microsoft.com/pubs/147663/TouchDevelop-MSR-TR-2011-49.pdf).

[12] *web*, WWW Document, (https://www.touchdevelop.com/docs/web).

[13] *platforms*, WWW Document, (https://www.touchdevelop.com/platforms).

[14] *Create and share*, WWW Document, (https://www.touchdevelop.com).

[15] *How fast is JavaScript compared to Java*? WWW Document, (http://stackoverflow.com/questions/3723374/how-fast-is-javascript-compared-to-java).

Appendix

A1. Android Toast Notification Example:

```
Toast.makeText(getApplicationContext(),
                    "Reloading web page",
                    Toast.LENGTH_LONG).show();
```

This code creates a toast notification in Android with the message « Reloading web page » with a length of « long ». It then displays the message on the screen with the method call show().

A2. Contact Adapter (extends ArrayAdapter):

```
package com.example.testapplication;

import android.app.Activity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import java.util.List;

public class ContactAdapter extends ArrayAdapter<Contact> {

   private final List<Contact> _contacts;
   private final Activity _context;

   public ContactAdapter(Activity context, List<Contact>
contacts) {
      super(context, R.layout.contactlistitem, contacts);
      this._contacts = contacts;
      this._context = context;
   }

   static class ViewHolder {
      protected TextView text;
      private Contact _contact;

      protected void setContact(Contact contact) {
         text.setText(contact.getDisplayName());
         _contact = contact;
      }

      protected Contact getContact() {
         return _contact;
      }
   }

   @Override
   public Contact getItem(int position) {
      return _contacts.get(position);
   }

   @Override
   public View getView(int position, View convertView,
ViewGroup parent) {
      View view = convertView;
      if (convertView == null) {
         LayoutInflater inflater = _context.getLayoutInflater();
         view = inflater.inflate(R.layout.contactlistitem, null);
         final ViewHolder viewHolder = new ViewHolder();
         viewHolder.text = (TextView)
view.findViewById(R.id.txtDisplayName);
         viewHolder.setContact(_contacts.get(position));
         view.setTag(viewHolder);
      }

      return view;
   }
}
```

A3. Processing Speed Code (shows logic similarity):

Native Android Application:

ProcessingSpeedActivity.java

```
package com.example.testapplication;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ProcessingSpeedActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.procspeed);

      ProcessingSpeed p = new ProcessingSpeed();
      long startTime = System.currentTimeMillis();
      p.calcRandomStuff();
      long endTime = System.currentTimeMillis();
      long duration = endTime - startTime;

      TextView textView = new TextView(this);
      textView.setTextSize(12);
      textView.setText("The calculations took " + duration + "
milliseconds.");

      setContentView(textView);
   }
}
```

ProcessingSpeed.java

```
package com.example.testapplication;

import android.app.Activity;

public class ProcessingSpeed extends Activity {

   private int numProcesses = 0;

   public ProcessingSpeed() {
      // empty default constructor as it is not needed (method is
directly called)
   }

   public void calcRandomStuff() {
      int x;
      int y;
      int z;
      for (int i = 0; i < 1000000; i++) {
         x = (int) (Math.random() * 11);
         y = (int) (Math.random() * 11);
         z = x * y;
         numProcesses++;
```

```
      }
   }
}
```

PhoneGap Application:

Processing.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>PhoneGap Testing Application</title>
      <meta name="viewport" content="width=device-width,
height=device-height, initial-scale=1, maximum-scale=1">
   </head>

   <body onload="onLoad()">
                                                <script
type="text/javascript">
         begintime = new Date().getTime()/1000

         for(i = 0; i<1000000; ++i) {
            a = Math.random() * 11
            b = Math.random() * 11
            x = Math.floor(a)
            y = Math.floor(b)

            z = x * y;
         }

         endtime = new Date().getTime()/1000
         speedtime = endtime - begintime
         speedtime = speedtime * 1000

         document.write("The calculations took " + speedtime + "
milliseconds.")
      </script>
   </body>
</html>
```

TouchDevelop Application:

Processing Script:

```
action main ()
   var dt1 := time → now → hour * 3600
   var dt3 := time → now → minute * 60
   var dt4 := time → now → second
   var dt5 := dt1 + dt3 + dt4
   var i := 0
   for 0 ≤ i < 1000000 do
      var x := math → random(11)
      var y := math→ random(11)
      var a := math→ floor(x)
      var b := math → floor(y)
      var c := a * b
   var dt2 := time → now → hour * 3600
   var dt6 := time→now →minute * 60
   var dt7 := time → now → second
   var dt8 := dt2 + dt6 + dt7
   var diff := dt8 - dt5
   diff → post to wall
```

A4. General Other TouchDevelop Code:

Open a Web page

```
action main ()
    web → browse("http://soe.rutgers.edu/gset")
```

Camera

```
action main ()
    media → choose picture → post to wall
```