An Assessment of Design and Implementation Trade-Offs and Their Impact on Mobile Applications

Nicholas Rogness
Department of Computer & Information Sciences
Minnesota State University, Mankato
nicholas.rogness@mnsu.edu
(507) 271-8136

Steven Case
Department of Computer & Information Sciences
Minnesota State University, Mankato
steven.case@mnsu.edu
(507) 389-5310

Abstract

When building an application for a mobile platform, there are many aspects to consider; portability, speed, and the user interface are just a few. Because there are several mobile devices that are capable of running custom applications and the costs associated with software development continue to increase, software portability is often a desired attribute of mobile applications. In addition, mobile devices are typically constrained with respect to memory and processor performance. Unfortunately, the effect of design and implementation decisions relative to these limitations will often conflict with the portability and usability of the software.

If an application is easy to move from one device to another, the application is considered portable and less time is needed to create releases for new devices as well as maintaining updates for all devices. However, the design and implementation requirements associated with portability often create conflicts with other desirable attributes of the application. Mobile devices tend to have limited computing power and memory relative to a conventional desktop, which requires developers to become more conscious of the processing and memory efficiency of their application. While limited memory and processing power can be viewed as similar problems between many of the mobile platforms, the user interface creates a new problem for developers. Whereas desktop environments generally have the same de facto layout, user interfaces for mobile devices vary significantly and impose additional constraints on software design and implementation

Leveraging from recent experience developing a commercial prototype of a language translation application, this paper will explore options the developer has to choose from in terms of programming languages and user interface design. The choice of implementation language can affect the way the application looks and performs. For instance, a Java application can be run under several different virtual machines, each making its own options and styles available to the developer. Although there is not one perfect design, techniques exist that make the development of a mobile application more efficient.

Introduction

Mobile devices have become increasingly popular over the last few years. Along with popularity, the variety of devices available to the general public has also grown significantly. Pocket PC, Palm Pilot, Smart Phone and Blackberry are a few names that have emerged in the mobile computer market. Each device has its own size, shape and style.

With a growing number of devices to support and an expanding consumer base, a problem that has challenged developers for years again comes into view. Software portability, a characteristic that allows an application to move from one platform to another with few, if any, changes, is becoming increasingly valuable. The motivation behind supporting software portability is lower cost of development. As the price of software development rises, it is becoming more and more valuable to reuse as much code as possible. This is not a trivial task for any software, but when one deals with mobile applications, there are more elements to consider than just differences in architecture and APIs.

Aside from the architectural aspect, having a wide variety of mobile devices brings another new twist to an old subject. Several designs of portable computers have evolved over the years. Some devices have a pen that takes place of a mouse with a virtual keyboard or some sort of hand-writing recognition, some have a full keyboard and touch screen, while others, such as the Smart Phones, have neither a pen nor a full keyboard. Just as the diversity of platforms creates issues with software portability on mobile devices, making an efficient and productive user interface for any particular device has its own challenges.

Performance is yet another key area that must be considered when developing for a limited resource environment. These considerations must not be taken lightly. Most users employ their devices during some other activity, such as a meeting. Most people are not willing to wait very long for an application to load. For a developer, this kind of programming does not come naturally, but if they remain in a state of mind that is ever watchful for wasted processing power, optimization will be easier to manage later on.

Portability and interface style are complicated enough problems to solve in their own right, and adding the strict performance requirements necessary for limited resource devices removes many solutions typically available to the developer. Decisions will have to be made about what to sacrifice. The most portable options generally have poor performance or irregular interfaces. Options that meet the conventional practices of the initial device, although they may meet performance requirements, may be complicated to move to another platform. These decisions need to be made based on the goal of the application, the typical users, and the resources available to the development team. Along with the analysis, extensive design is required to ensure the application is developed in such a way that the maximum amount of code can be reused. In some situations, the number of lines of code reused can serve as a metric to analyze the development process itself.

This paper will examine the different options a developer has to choose from when designing and coding an application targeted to a mobile platform along with the tradeoffs that come with each option. There is no one option that is inherently better than another, but a sample application will be used to show that in a particular instance, some coding methods are not feasible.

Other Research

There has been much research in the areas of portability and usability. The majority of this research has been centered on computers that are more powerful and more similar than the devices this paper will focus on. This information is provided as a background for details mentioned later on. Much of the outlined strategies can be applied to mobile applications and are useful in identifying complications that may occur during development.

Application Portability

James D. Mooney has written several pieces on the topic of software portability. In one such paper (1992), he outlines a course whose object was to instruct the students on how to write code that is to run on several different platforms. The class used a simple interactive quiz application written for one platform. The goal was to identify the obstacles they would encounter when porting the application and make changes to it that would make converting it for another platform simpler. Functional enhancements were added to the program later on to create even more portability issues. Even though it was a command line interface, not every system had the same conventions for entering parameters. One of the issues discovered was that an application that is portable doesn't necessarily have to behave in the exact same way for every platform. Such issues are clearly evident in mobile devices where each platform has developed its own look and feel. Any changes in appearance and behavior are easily noticeable to the user. Not only are the changes noticeable, but they also may give the user a negative view of the application.

In 1995, Mooney wrote a paper that discusses issues concerning software portability. Along with that topic, he compares software portability to software reusability. Both of these are interesting ideas when developing mobile applications. Mooney states, "The goal of research in software portability is to facilitate reuse of existing applications in new environments." Reuse has a similar goal, but has a different scope in terms of the project. Reuse is more concerned with reusing components in several applications, but not as concerned with which platform the software is running on. Mooney goes on to describe concepts unique to each school of thought.

The general term, portability, is used to describe a characteristic that allows an application to be ported. This can be achieved in two ways. Binary portability is the

ability to move the binary executable file to another platform. This is by far the optimum choice, but is only possible in a few situations. The other is source portability, which describes code that must be moved to the other environment and rebuilt, which is more forgiving to the developer. With this as an acceptable means, developers have the freedom to make changes to the code depending on what environment a particular version will be running on. Even though source portability allows some code to be changed, the goal remains to change as little code as possible. In an earlier article, Mooney (1990) mentions a third, and least optimal solution: experience portability. This does not concern moving code from one environment to another, but instead focuses on the porting of ideas and algorithms. If the source code for a component has to be rewritten, the developers can use the same design and implementation techniques as the original. This may not save actual coding time, but doesn't require the developers to start from scratch either. UML and code generation tools could be used to aid the developers in rewriting certain functions.

Reuse has a separate focus from portability. Instead of moving an application from one environment to the other, reuse concentrates on modularizing applications so the pieces developed for one project can be used in another. From a management perspective, much is gained from having a repository of reusable components (Mooney refers to them as artifacts). When a new project is being designed, developers can turn to such a repository to find functions, data structures, and other segments that can be plugged in.

Mooney compares and contrasts these two concepts in such a way that one can be described in the terms of the other. There us much common ground between these, but he does point out that portability has benefits that may not be seen immediately, while reusability has its advantages during the early stages of development. There are ways in which to incorporate reusability techniques to make porting an application easier. When dealing with the variety of styles in the mobile device market, modularity of code may become useful for creating a suite of products to cover each device.

Interface Design

Although most of the topics involved in Human Computer Interaction are beyond the scope of this paper, it is beneficial to look at a few of the concepts which help us make decisions on what elements are necessary in a user interface and which ones can be sacrificed for other benefits.

Most developers learn to write code for either a personal computer or mainframe, depending on the time and place where they start. Because of this, it is not always easy to shift one's thoughts on what a portable application should look like. Mark Dunlop gives a good overview of the design considerations for mobile applications that do not come naturally for most developers (2002). First, application developers must consider the typical environment the user will be in when accessing the software. If it resides on a PDA, the user could be almost anywhere and not have access to normal office items, such as a desk. Generally, mobile applications will be used by a wide variety of people who

may not have any formal training. Most of these small devices have cumbersome, if any, input devices; therefore, it becomes ideal to let the user type as little as possible. Finally, due to the quick and easy nature of mobile devices, frequent interruptions must be expected. The user may want to turn the device on, extract a small piece of data, and turn it off without spending much time waiting for a process to finish.

Findings

As with many lessons, some are learned as they are needed. Based on the recent development of a language translation application for mobile devices, several concerns have become apparent along with their possible solutions. The problems found vary from performance to behavioral. For the application, some short-term solutions have been implemented, but they are far from optimal.

From research gathered and experience gained from development, better solutions are available. None of these are perfect, so both benefits and trade-offs will be examined.

Goals of portability for mobile devices

Aside from goals of a specific software application, there are some characteristics that are important to all portable, mobile applications. Just as with any portable application, it becomes vitally important to maximize the amount of movable code. Somewhat unique to limited platforms is the emphasis on the conservation of memory and processing power. Behavior is a factor that most developers would not normally consider when writing an application for one device. When targeting a wide range, it is important to note that different user interfaces may be required for each device.

Maximize movable code

Mooney described three types of portability that can be used to minimize the amount of rewritten code. Although he may not have been considering mobile applications, his concepts can still be applied.

Binary portability is certainly the most efficient method of portability. If it is possible to move an executable program from one platform to another, porting becomes a non-issue. Unfortunately, it is also the most difficult, and many times impossible, to achieve. Even when limiting the scope of target devices to one manufacturer and processor, factors such as screen size and input devices cause the need for separate versions of the software.

Source portability comes closer to the realm of possibility; however there are some issues that can make porting source code a difficult process. Some standard functions, such as string functions, are included in most development tools, but are represented by different names. There are several ways by which adapting the code can be made easier. Macros

and other compiler directives can be used to substitute function names. For standard functions and basic file access or calculation functions, these strategies may work well, but when calls to the operating system or graphical user interface are required, much of the code may not be directly portable. In these cases, a better way to think of portability is in terms of experience portability.

Even though it is likely that individual user interfaces will be needed for each device, most of the ideas can be reused. The overall design of the interface can remain, for the most part, unchanged. Interfaces to other components or the application that were more portable should remain the same as well.

Performance

Performance is another aspect that can be a considerable factor in the in the success of a portable application. Mobile devices are commonly used for very short-term functions, such as storing a phone number or adding a calendar entry. This adds a requirement that the application be able to load as soon as the user needs it, and execute just as quickly. If the user becomes annoyed by the time it takes to run the application, they will stop using it.

Memory can create similar problems on mobile devices. Because several applications can be running in the background at the same time without the user realizing it, if one program is consuming the majority of the system's resources, other applications will suffer. The easiest solution for the user is to no longer run the program that is slowing everything else down.

Behavior

Finally, behavior can have just as much effect on the user's experience as performance. Over the years, mobile devices have created their own styles that most applications conform to. Noticeable deviations from such conventions can make the program seem out of place and cumbersome. Some designers may desire to stray from these, but for this paper, the assumption is that the program isn't meant to break down any usability barriers. Even though this consideration may contradict portability, making a usable and robust application may be a higher priority.

The very nature of these goals does not allow them to be implemented perfectly however. Due to this fact, decisions have to be made early on, about which goals are more important. These decisions should be based on the intended use of the application.

Case Project

Many of these lessons concerning portability of mobile applications came from the actual development of a commercial language translation application. Because the details of this specific application are not relevant, it will be referred to as Ignotus. Several goals were laid out during the initial design stages. From these goals, the design team decided on an opening configuration, which was to be developed using the Pocket PC platform. This decision was made because of the resources available and initial users had most interest in the Pocket PC platform. During development, several problems arose which required the development team to make changes to the overall plan for the application.

The first and most important goal for Ignotus was portability. The designers wished to eventually move the application to other mobile devices. Secondly, the software must be easy to use by a typical owner of the device. The target users for Ignotus were people who are familiar with the device and are looking for new applications. It was believed that most people would purchase the product as an impulse and continue using it based on its functionality.

Implementation Options

When developing mobile applications, there are essentially three choices for languages, C++, Visual Basic, and Java. Visual Basic is not an option in this case since the only platforms that support VB are Microsoft based. Using VB in an application would extremely complicate the porting process so it is safe to remove it from the list of options. Depending on the device and available tools, others may be available, but for porting purposes, C++ and Java are the most widely available. Using these languages, there are three options available to developers of an application such as Ignotus. The program can be written completely in Java, completely in C++, or certain techniques can be used to combine the two languages.

All Java

The obvious advantage to developing in Java is also its main selling point, portability. Ideally, developers can write code for one platform and the same code will work on any other device that has a JVM. This quality makes for faster development time since little or no code needs to be modified when porting the application.

Java adds another category of portability. It cannot be correctly labeled as binary portable because it is not compiled down to a binary format; instead it uses interpreted byte code. It is also different from source portable in that code should not have to be changed and only needs to be compiled once. Even though it does not fit within a class described by Mooney, there are some situations that would make it fit better in certain groups.

Theoretically, changes to the application shouldn't be needed to move it from one platform to another. Even with Java, this is not always the case. The first problem comes from the JVM itself. Unlike the desktop versions, Sun does not have one set JVM specification for standard mobile applications. There are several different specifications that a JVM can conform to (i.e. J2ME CDC, J2ME CLDC and Personal Java). Since the same virtual machine may not be available on every device, the developer is either forced to use only classes that are shared between the different platforms, or develop the application for one specification, and then port it to another. Fortunately, the highly object-oriented design of Java makes it easier for developers to write code this way using modularization. If designed well, components should have the ability to be added and removed without disrupting the program's logic. Certain classes such as those in windowing toolkits may not be universally supported. To make porting simpler, the developer can use abstract classes and interfaces to minimize the amount of code that needs to be changed when a different toolkit is used.

Adding some extra steps in the design stage, and researching what classes will and won't be supported can result in minimal code changes when moving the application from one environment to another. When designing an application to be developed in Java, more about the target environments must be known to make full use of Java's advantages.

One of the constant issues with these small devices is execution time and memory consumption. Performance has been a considerable complaint about Java. This is an inherent problem since Java byte-code needs to be interpreted each time the application is executed. This can consume precious clock cycles on a machine that is not very powerful to begin with. Certain functions can especially cause a decrease in the performance of an application. String concatenation, along with other string functions, has been known to be a complex operation in Java. Limiting calls to these kinds of functions can increase the overall speed of the application.

Memory usage is also a topic that developers need to keep in mind while writing code. Even though Java has a garbage collector, the programmer must stay vigilant against memory leaks. Static objects and data structures can take up much of the limited resources. For the most part, these issues can be addressed by good programming skills and techniques. It is far more difficult to address memory usage after the application has been developed than for the developers to constantly consider how they can use fewer resources while writing the application code.

An issue that is dependent on the JVM itself is window design and behavior. Unfortunately, the Java specification defines what classes and methods must be implemented, not how they are implemented.

The developers of Ignotus ran into these types of problems when developing their application. The software required numerous accesses to a file, which was discovered, could also be a costly operation. After some testing and experimentation, they found that the implementation of the file input/output classes was not sufficient for their needs. Instead of using the standard classes supplied with the JVM, the developers began to

write their own classes to optimize the functions they were most interested in. This was added effort, but in the end, improved their performance considerably. In such a limited environment, creative programming techniques are often required. Programming in Java for a mobile device is significantly different from developing such an application for a desktop machine.

When researching a JVM for Ignotus, the developers found that no two virtual machines displayed a window in the same manor, nor was one virtual machine available for all of the target platforms. Of these window styles, none matched the conventions that have developed in the Pocket PC market. The designers of Ignotus felt that this would be a strong disadvantage for the application. A program that does not look like it belongs on a Pocket PC may give the user a negative impression before they even start using any of the functions.

Figures 1 and 2 display applications running on a Compaq iPaq. The screenshots were obtained using Remote Display Control available from Microsoft. Figure 1 was generated using Microsoft Embedded C++ and shows the common layout of a Pocket PC application written in C++. Figure 2 was written using Java and run with a JVM compliant with Sun's PersonalJava Specification. Several differences can be seen. First, the Start Menu has moved from the upper left, to lower left. Second, the keyboard button has moved from the right side to the center. In this case, the Java application in Figure 2 acts more like a Windows application than its counterpart. The round 'X' button in Figure 1 does not actually exit the application. Instead, it remains running in the background. The next time the application is executed, the window manager merely brings it to the front of the screen. The buttons in the upper right corner of Figure 2 act similarly to a Windows application, although functions such as minimize and maximize loose their usefulness in such a small screen space. One distinct advantage Java has over C++ is the amount of code required to develop such and application. The source code generated for Figure 1 is about 200 lines. Figure 2 required a tenth of that.

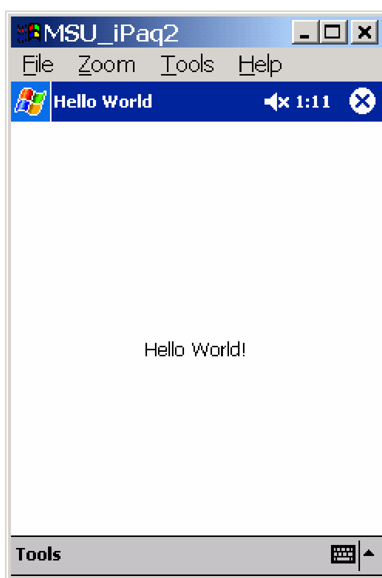

Figure 1: C++ Application

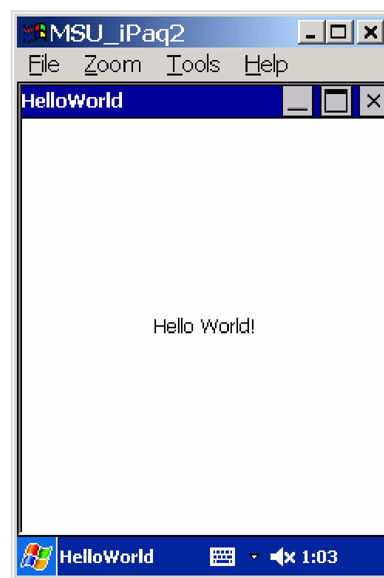

Figure 2: Java Application

Due to the problems with Java, the Ignotus team decided to discontinue their development of the application using Java. A functional prototype was developed using Java, but was far from ideal in both performance and behavior. However, for some applications that are more concerned with portability and fast deployment, Java may be the best option.

All Native

Almost a complete opposite from the Java approach to building an application is to develop the program using C or C++. Developing with C++ can have many advantages to Java, but there are some issues which can make it much more difficult to work with.

Where Java is built to be completely portable, C++ is not. Almost any compiler you find for each device will support standard C and C++ libraries, but that does not necessarily mean you can just recompile the source code and it will work. One of the issues concerning portability is the difference in function names. Some of the functions in the Palm OS standard C libraries have been changed to match the Palm OS function-naming scheme. These kinds of issues can be solved by macros, or work-around functions. Adding functions just to redirect calls to the correct names can increase the size of the executable. This is not desirable when dealing with limited storage spaces.

The subject of function names only applies to the standard C libraries. Other libraries used for windowing and OS functions will not be common between devices. A discussion of these differences is beyond the scope of this paper, but such information is readily attainable. The fact that there is a significant difference between mobile operating systems limits the degree to which the applications can be portable. One possible solution to this problem would be to use one of the reusability techniques mentioned by Mooney, modularization. This was mentioned early when discussing Java programming, but can be applied here as well. In this case, it would need to be used to a greater degree, but the purpose is still the same. Although it is a large amount of code, the idea is to share code between versions so that the amount of code that is rewritten is kept to a minimum. This can be implemented by using standard C libraries for logic and algorithm functions that will perform the same task throughout each version. User interface and OS specific functions should have a standard interface with other components to facilitate easy removal when needed.

Performance is one of the first reasons to develop an application in C or C++. Because the code is compiled down to machine language, it will be able to run without the need for an interpreter. This also gives the programmer the ability to use operating system functions to gain access to system resources, such as files and hardware. Because the functions don’t require generality to work in any environment, they will be optimized to work specifically for the target device.

Along with performance, behavior is another advantage that C++ has over interpreted languages. The purpose of portability is not necessarily to make an application look and act exactly the same on every platform (Mooney, 1992). Due to conventions used by applications written for a specific device, a program written in a native language would have an advantage over a program meant to be universal. With an interface that follows the de facto standards or a device, the user will not be distracted by an uncommon style. Instead, the user can focus on functions of the application and its usefulness.

Even though native languages have their advantages over interpreted languages, it is not always the right choice. Depending on the application, the performance and behavior differences may not be significant enough to sacrifice portability.

The Ignotus team was hesitant to accept this option as the final one. Due to increased development time and inexperience on the part of the programmers, other options were explored before C++.

Combination

If the design obligations do not lend themselves to developing the application in all native code, or all interpreted, such as Java, a combination of each could be used. Using Java's JNI technology, it is possible to combine the portability of Java with the performance and behavior characteristics of ++. The Java Native Interface is well documented, but does increase the programming complexity significantly.

The idea behind this concept is to combine the advantages of each technology. This should create an application that is both fast and relatively portable. To make all of this work, the developer is required to maintain an interface that is consistent between native and Java code.

The Java must be developed first. Any function that will be implemented using a native language must be declared in Java using the native keyword.

```
public native void HelloWorld();
```

After it is compiled, the developer uses the javah tool with a –JNI option to create a header file for the class. This will notify the native compiler about the functions available to be called. The native code is then compiled into a DLL or some other library, depending on the platform. When the Java application is executed, the JVM loads the library specified. Execution begins in Java. Any time a Java method calls a native function, the execution path is shifted to the native library. When the native function returns, execution continues in Java.

When dealing with such a complex technology, it becomes more difficult to maintain continuity between languages. Calling Java methods from a native function requires the exact name and class that describes it along with the correct type and number of

parameters. Java data types are consistent from one platform to another, but the size of native data type can vary depending on the device. These complications make using JNI much more problematic than developing the entire application in one language.

Because of performance problems with Java, the Ignotus team briefly explored the possibility of using JNI to achieve their goals. However, the team eventually decided to forgo this route. Problems arose due to the JVM that was being used for development. Being that it is a commercial product, it was not easy for the developers to obtain specific information concerning how some features were implemented. For a C++ function to display a user interface on a Pocket PC device, it must have certain variables normally passed to the initial function when the operating system executes the program. These variables are note available when the functions are called from Java. Conversely, library files are needed for a native function to initialize the JVM and begin using Java classes and methods. Unfortunately, these libraries were not attainable by the developers.

Depending on the nature of the interface, this process can become very involved. Strict attention by the programmer must be maintained when modifying code on either side of the JNI functions. To determine whether all of this added work and complexity is acceptable, performance testing should be done to aid the designers in making the decision.

One More Solution

Similar to combining Java technology with native programming languages is the option of creating a custom JVM. This would require a large amount of work for the initial project, but will decrease the amount of time needed for future projects. JVMs developed by third-party vendors are developed with the idea that it should be used for any application any Java developer decides they want to write. If a company were to develop its own JVM, this universal use requirement would no longer apply. Each class could be developed with the specific project in mind. Each of the functions could be optimized to meet the performance requirements specified by the designers. Instead of creating a JVM that implements every function for every class laid out by the JVM specification from Sun, only the classes and functions necessary for the project need to be implemented. Although, this mini JVM will not be compliant with Sun's documentation and therefore could not be officially referred to as a JVM. As a future goal of the company, a JVM that does meet Sun's requirements could be developed and distributed.

All of this, of course, is not easily accomplished. This kind of technique requires the developers to have resources and ability to create such a tool. Sun has reference implementations and the corresponding source code available for two of its J2ME specifications for mobile devices. The Connected Limited Device Configuration (CLDC) specification is meant for devices with limited memory capacities such as cell phones and some pagers. Connected Device Configuration (CDC) applies to devices with more than 2MB of memory. Either of these could be ported from their reference implementation to the target platform.

If the company has already developed a JVM, it can be modified where needed for future projects. Some modifications may be necessary and functions may need to be added. A larger task would be porting the JVM when another platform is added to the company's repertoire. Since several teams working on several projects may use the program, a developer or team of developers may be needed to maintain the JVM and oversee any changes made to it.

Conclusion

The options available to developers of mobile applications go far beyond those outlined in this paper. The creativity of programmers and software designers opens a door to infinitely many possibilities. Standards for development of mobile applications have been delayed for this very reason. Limiting what developers can invent by making them conform to guidelines can only harm the result. There is no way to tell what future programs will look like or how they will be developed. For now, programming methodologies that have been proven in the desktop work are being applied to mobile applications despite the enormous differences in performance and requirements. When developing a mobile application, an alternate mindset is required. Thinking as a desktop programmer produces code that works perfectly on a desktop, but when executed on a mobile device will not produce a desirable application.

Creating a robust and complete mobile application requires a clear understanding of the application itself, and what is required of it. As with other software applications, spending time in the design phase to determine the correct options for a program will decrease the amount of time needed to write the actual code. Viewing the application as a combination of its requirements and functionality will give the design team information and insight to make decisions about the product that will save development time and maximize usability.

References

1. Mooney, James D. (1990). Strategies for Supporting Application Portability. *IEEE Computer*, 23, 59-70.

2. Mooney, James D. (1992). A Course in Software Portability. *ACM SIGSCE Bulletin,* 24, 53-56.

3. Mooney, James D. (1995). Portability and Reusability: Common Issues and Differences. *Proceedings of the 1995 ACM 23rd annual conference on Computer science*, 150-156.

4. Dunlop, Mark, & Brewster, Stephen. (2002). The Challenge of Mobile Devices for Human Computer Interaction. *Personal and Ubiquitous Computing*, 6, 235-236.